Mobile DLS

(Downloadable Sounds Format for Mobile Applications)

Version 1.0a

November 2004

Including dls.h Corrections

Published By:

The MIDI Manufacturers Association

Los Angeles, CA

Preface

This Specification document contains the complete description of Mobile DLS. Since Mobile DLS is derived from DLS 2.1, this document is largely identical to the DLS 2.1 Specification document but specifies where Mobile DLS deviates from the DLS 2.1 specification. Readers should not need to refer to the DLS 2.1 Specification to understand Mobile DLS.

The MMA and all companies involved in the development of DLS-2 shall not be held liable to any person or entity for any reason related to the adoption, implementation or other use of this document or the specification herein.

Changes for version 1.0a:

- Removed: "`ULONG ulEntryIndex; /* Index entry in the list */`"
  from: "`typedef struct _POOLCUE {`"
  in file: "dls.h".
- Corrected letter case in: "`#define FOURCC_wave mmioFOURCC('W','A','V','E')`"
  in file: "dls.h".

Mobile DLS Specification

RP-041

Copyright © 2003/2004 MIDI Manufacturers Association Incorporated

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.

First Printing 2004

MMA
PO Box 3173
La Habra CA 90632-3173

TABLE OF CONTENTS

1. Synthesis Model 1
1.1 Abstract 1
1.2 Overview of Synthesizer Model 1
1.2.1 Control Logic 2
1.2.2 Digital Audio Engine 2
1.2.3 Articulation Modules and Connections 2
1.2.4 DLS Models 3
1.3 Minimum Device Requirements 5
1.4 Synthesizer Design Detail 5
1.4.1 Control Logic 5
1.4.2 Articulation Data 6
1.4.3 Conditional Chunks 7
1.4.4 Note Exclusivity 7
1.4.5 Voice Allocation 7
1.4.6 Bank Select and Program Change 8
1.4.7 Phase-Locked Samples 8
1.5 Digital Audio Engine 9
1.5.1 Digital Oscillator 9
1.5.2 Digitally Controlled Filter 10
1.5.3 Digitally Controlled Amplifier 13
1.6 Articulation Modules and Connections 13
1.6.1 Signal Flow 13
1.6.2 Connection Block 14
1.6.3 Default, Global, and Local Articulation Data 15
1.6.4 Controller Theory 15
1.6.5 Transforms 16
1.7 Generators 19
1.7.1 Low Frequency Oscillator 19
1.7.2 Envelope Generator 19
1.7.3 Key Number Generator 23
1.8 Performance Controllers 23
1.8.1 Note Number 23
1.8.2 Volume 23
1.8.3 Expression 23
1.8.4 Velocity 23
1.8.5 Pan 23
1.8.6 MIDI Controller 1 (Modulation) 24
1.8.7 Channel Pressure 24
1.8.8 Pitch Bend 24
1.8.9 MIDI Controller 64 (Sustain) 25
1.8.10 MIDI Controller 91 (Reverb) 25
1.8.11 MIDI Controller 93 (Chorus) 25
1.8.12 Registered Parameter Numbers (RPN) 25

1.8.13 Registered Parameter Number Data Entry 25
1.9 Channel Mode Messages 26
1.9.1 Reset All Controllers (MIDI Controller 121) 26
1.9.2 All Notes Off (MIDI Controller 123) 26
1.9.3 All Sound Off (MIDI Controller 120) 26
1.9.4 Other Channel Mode Messages 26
1.10 Active Sensing (0xFE) 27
1.11 Default Values 27
1.12 Articulation Architecture 27
1.13 Modulation Routing 27
1.13.1 Default Connections 27
1.14 Data Format Definitions 32
1.14.1 Absolute Pitch 32
1.14.2 Relative Pitch 32
1.14.3 Absolute Time 32
1.14.4 Gain 32
1.14.5 Sample Frequency 32
1.14.6 Instrument 32
1.14.7 Region 33
1.14.8 Wave Link 33
1.14.9 Articulation 33
1.14.10 Wave Sample 34
1.15 Tolerances 34
1.15.1 Digital Oscillator 34
1.15.2 Digitally Controlled Filter 34
1.15.3 Digitally Controlled Amplifier 35
1.15.4 LFO Generator 35
1.15.5 Envelope Generator 35
1.16 DLS System Exclusive Messages 36
2. DLS File RIFF Structure 37
2.1 RIFF Format 37
2.2 RIFF Structure 37
2.3 LIST Chunk 41
2.4 <colh-ck>, Collection Header Chunk 41
2.5 <dlid-ck>, DLSID Chunk 41
2.6 <cdl-ck>, Conditional Chunk 43
2.6.1 DLS-1 and DLS-2 Compatibility 45
2.7 <insh-ck>, Instrument Header Chunk 46
2.8 <rgnh-ck>, Region Header Chunk 47
2.9 <art1-ck>, DLS-1 Articulator Chunk 48
2.10 <art2-ck>, DLS-2 Articulator Chunk 50
2.11 <wlnk-ck>, Wave Link Chunk 54
2.12 <wsmp-ck>, Wave Sample Chunk 55
2.13 <ptbl-ck>, Pool Table Chunk 56
2.14 <vers-ck>, Version Chunk 57
2.15 <INFO-list>, INFO List Chunk 58

2.16 DLS Wave File Format 60
2.16.1 Format Chunk <fmt -ck> 60
2.16.2 Data Chunk <data-ck> 62
2.16.3 Data Packing for WAVE_FORMAT_PCM Files 63
2.16.4 Data Packing for 8-Bit Mono PCM 63
2.16.5 Data Packing for 8-Bit Stereo PCM 63
2.16.6 Data Packing for 16-Bit Mono PCM 63
2.16.7 Data Packing for 16-Bit Stereo PCM 63
2.16.8 Data Format of the WAVE_FORMAT_PCM Samples 63
2.17 Instrument Object Hierarchy 64
2.18 Proprietary Chunk IDs 65
2.19 File Examples 66
2.19.1 Generic DLS-1 File 66
2.19.2 DLS-1 File With 3rd Party Extensions 68
2.19.3 Generic DLS-2 File 70
3. Compatibility Notes 72
3.1 Coding Requirements and Recommendations 72
3.2 DLS-1 and DLS-2 Compatibility 72
3.3 DLS-1 Compatibility 72
3.4 Mod LFO to Gain Change (DLS 2.1) 73
3.5 DLSID Integrity 73
4. DLS Header Files 75
4.1 DLS-1 Header File 75
4.2 DLS-2 Header File 82
4.3 Mobile DLS Header File 86
5. References 87

FIGURES

FIGURE 1 — BLOCK DIAGRAM OF MOBILE DLS SYSTEM ARCHITECTURE....4
FIGURE 2 — CONTROL LOGIC....6
FIGURE 3 — FORWARD LOOP....9
FIGURE 4 — LOOP AND RELEASE....9
FIGURE 5 — NON-RESONANT LOW-PASS FILTER....11
FIGURE 6 — RESONANT FILTER....11
FIGURE 7 — FILTER CUTOFF FREQUENCY....12
FIGURE 8 — MOBILE DLS CONNECTION BLOCK....14
FIGURE 9 — TRANSFORMS....17
FIGURE 10 — ENVELOPE....20
FIGURE 11 — PAN CURVE....24
FIGURE 12 — DLS FILE FORMAT....40
FIGURE 13 — USTRANSFORM FIELD....51

TABLES

TABLE 1 — COMPARISON OF DLS MODEL MINIMUM REQUIREMENTS....5
TABLE 2 — SUPPORTED NOTE-EXCLUSIVITY FUNCTIONALITY....7
TABLE 3 — SUPPORTED VOICE-ALLOCATION FUNCTIONALITY....8
TABLE 4 — SUPPORTED DIGITAL OSCILLATOR FUNCTIONALITY....10
TABLE 5 — MODULATION ROUTING....28
TABLE 6 — DEFAULT CONNECTION BLOCKS....30
TABLE 7 — RIFF DEFINITIONS....39
TABLE 8 — DLS-1 SOURCES, CONTROLS, DESTINATIONS, AND TRANSFORMS....50
TABLE 9 — DLS-2 SOURCES, CONTROLS, DESTINATIONS, AND TRANSFORMS....52

1. Synthesis Model

1.1 Abstract

The prevalence of synthesizers in devices intended solely for consumption of synthesized audio has created the need to standardize instrument formats. Although the General MIDI (GM) specifications [4,5] for instrument presets help to a degree, they lack the depth and breadth to deliver a truly consistent playback experience across a wide range of platforms. There is a need for a musical instrument standard that allows composers to define exactly how each musical instrument sounds on a wide variety of playback devices.

In January 1997, the MIDI Manufacturers Association ratified the first industry-wide standard for transporting complete sounds between synthesizers. The Downloadable Sounds Level 1 Specification (DLS-1) [2] defines a first order cross-platform synthesizer model that allows developers to target a large range of playback devices with a single set of content while still maintaining consistent playback across all those platforms.

In 2000, the Downloadable Sounds Specification Level 2 (DLS-2) [3] was introduced as an extension of DLS-1. The DLS-2 specification described a new playback device architecture as well as extensions to the DLS-1 file format. The DLS-2 specification was the result of collaboration between numerous hardware and software manufacturers represented by the MMA, as well as input from other organizations intending to use the format, such as MPEG (IEC/ISO JTC-1 SC-29 WG-11).

This Mobile DLS Specification defines a wavetable instrument representation format optimized for use in mobile applications. Technically Mobile DLS is based on the DLS-2 file format and supports both DLS-1 and DLS-2 technologies through the use of mechanisms introduced in the DLS-2 specification.

Mobile DLS departs from previous DLS specifications in a few notable ways, including:

- The technical distinction between melodic programs and percussion programs no longer exists. There is now a single bank + program space, and any bank + program address can hold either type of program. (ulBank bit 31 no longer treated as part of bank + program address.)
- Wavetables are no longer limited to 16-bit or 8-bit linear PCM. There is an extensible mechanism allowing the use of other encodings besides linear PCM, including compression codecs.
- It is possible to implement a Mobile DLS device without DCF and Vibrato LFO, however such an implementation will not be able to play instrument content that is authored to require those features.
- Minimum device requirements including polyphony and system sampling rate have been relaxed due to the limitations of mobile devices. Reverb and chorus have been made optional, and all voice allocation decisions are left to the implementor.
- Harmonization with SP-MIDI SMF content, including a requirement for a General MIDI or General MIDI 2 instrument set.
- Harmonization with Mobile XMF compound content (SP-MIDI SMF plus Mobile DLS instruments).
- Several corrections to the DLS 2.1 text in regard to the DCF, connection transform curve specifications, note exclusivity, and other matters, which subsequently should be adopted for DLS 2 as v2.2.

1.2 Overview of Synthesizer Model

A musical instrument definition in DLS is much more than a simple collection of audio samples. In addition to the actual sample data and associated loop information, the instrument must indicate under what circumstances each sample should be used, and how to modulate, or articulate, the sample as it plays.

[TW1]The typical synthesizer architecture can be broken down into three distinct subsystems:

- Control logic
- Digital audio engine
- Articulation modules and connections

1.2.1 Control Logic

The control logic receives a MIDI note event and determines which instrument should play the note, and, within that instrument, which sample and articulation combination to use.

Choosing the instrument requires little more than observing the MIDI channel number in the event and selecting the proper instrument accordingly.

Choosing the sample and articulation to use is not as simple. Almost all sampled synthesis architectures employ some method of organizing samples by note range across the keyboard. In addition, individual samples can be assigned to different velocity ranges or multiple samples can be played at once to create a richer, layered sound.

Terms such as layers, splits, blocks, and regions are commonly used in synthesizer jargon to refer to the management of multiple samples. For the purposes of this specification, they are referred to as *regions.* Regions are described further in the Synthesizer Design Detail section (Section 1.4).

1.2.2 Digital Audio Engine

The digital audio engine is the most obvious part of the synthesizer. It is composed of a playback engine (a digital oscillator), a digitally controlled filter, and a digitally controlled amplifier.

The digital oscillator plays a sampled sound waveform, managing loop points within the waveform so the sound can play continuously if needed. As the note plays, it can respond to changes in pitch, allowing for real-time expression such as vibrato and pitch bend.

The digitally controlled filter is a two-pole low-pass filter with dynamically controlled cutoff frequency and a programmable resonance. Typical applications for the filter include varying the amount of high frequency content according to MIDI velocity or applying the modulation envelope to reduce the upper harmonics over time to simulate acoustic instruments.

The digitally controlled amplifier modulates the volume of the instrument. Most importantly, this is used to control the amplitude shape, or envelope, of the note. It is also used for other types of real-time expression, such as tremolo.

Pitch, filtering , and volume control of the oscillator and amplifier are critical because they define the shape of the sound as it plays, and allow it to be dynamically responsive in real time, giving the sampled instrument much more expression than simple digital audio playback could ever provide. Real-time control of these parameters comes from modules in the Articulation section, which generate a constant stream of changing pitch, filter, and volume messages to which the digital audio engine responds.

The digital audio path represents the journey the sound takes from the oscillator to the filter, then to the amplifier, and then to the digital-to-analog converter (DAC). This path can optionally include additional modules, such as effects devices, that process the sound as it flows from oscillator to DAC.

1.2.3 Articulation Modules and Connections

The articulation modules are a set of devices that provide additional control over the pitch, filtering, and volume of the sample as it plays.

The articulation modules include low frequency oscillators (LFOs) to contribute vibrato and tremolo, envelope generators to define an overall volume and pitch shape to the sample, and MIDI real-time controllers, such as Pitch Bend and Modulation controller, to give the music real-time expression.

Generally, these modules can be linked in different ways to provide different results. For example, an LFO might generate a sine or triangle wave, which modulates the pitch of the sample for vibrato or the volume of the sample for tremolo. Modules can also receive as well as send control signals. An envelope generator might use key velocity to influence the attack time of the envelope or the cutoff frequency of the low-pass filter.

Articulation modules can be configured in different ways, and this configuration is an important part of the instrument definition. In fact, the term *patch*, as used for an instrument preset, refers to the early days when the hardware modules in an analog synthesizer were "patched" together with cables, which routed signals from module to module.

The ability to configure the modules is important because it yields a flexible approach to synthesizer design. At the same time, it is important to define a base-level configuration that can be supported by all hardware. This specification maps the routing onto a flexible system, allowing for future expansion to encompass not only extensions of wavetable synthesis, but other forms of musical synthesis as well.

1.2.4 DLS Models

1.2.4.1 DLS-1 Model

The DLS-1 synthesizer consists of the following basic elements for each voice:

- A sampled sound source with loop points (forward loop)
- Two 4-segment envelope generators characterized as ADSR (Attack-Decay-Sustain-Release)
- One Low Frequency Oscillator (LFO) generators
- Standardized response to MIDI controllers

1.2.4.2 DLS-2 Model

The DLS-2 synthesizer consists of the following basic elements for each voice:

- A sampled sound source with Loop and Release
- Two 6-segment envelope generators characterized as DAHDSR (Delay-Attack-Hold-Decay-Sustain-Release)
- Two Low Frequency Oscillator (LFO) generators
- A low pass filter (DCF) with resonance and dynamic filter cutoff frequency
- Standardized response to MIDI controllers

1.2.4.3 Mobile DLS Model

The Mobile DLS synthesizer consists of the following basic elements for each voice:

- A sampled sound source with loop points (forward loop)
- Two 4-segment envelope generators characterized as ADSR (Attack-Decay-Sustain-Release) corresponding to 6-segment DAHDSR envelope generators with the Delay and Hold set to zero.
- A modulation Low Frequency Oscillator (LFO) generator
- Standardized response to MIDI controllers

Optionally, the Mobile DLS synthesizer voice can contain the following elements:

- The sampled sound source may include Loop and Release in addition to forward loop
- Two 6-segment envelope generators characterized as DAHDSR (Delay-Attack-Hold-Decay-Sustain-Release) replacing the mandatory 4-segment envelope generators
- A vibrato Low Frequency Oscillator (LFO) generator in addition to the modulation LFO
- A low pass filter (DCF) with resonance and dynamic filter cutoff frequency

If a device supports any of the above optional features, they must all be supported.

The existence of the optional elements in content is indicated by a signalling mechanism (see Section 1.2.4.3.1).

***Note:* Optional features are not always referred to as such within this specification.** For example, the specification may simply describe two LFOs instead of distinguishing between the mandatory modulation LFO and the optional vibrato LFO. In a similar way, the specification may refer to the Loop and Release, 6-segment envelopes, and DCF without mentioning that they are optional for a Mobile DLS synthesizer implementation.

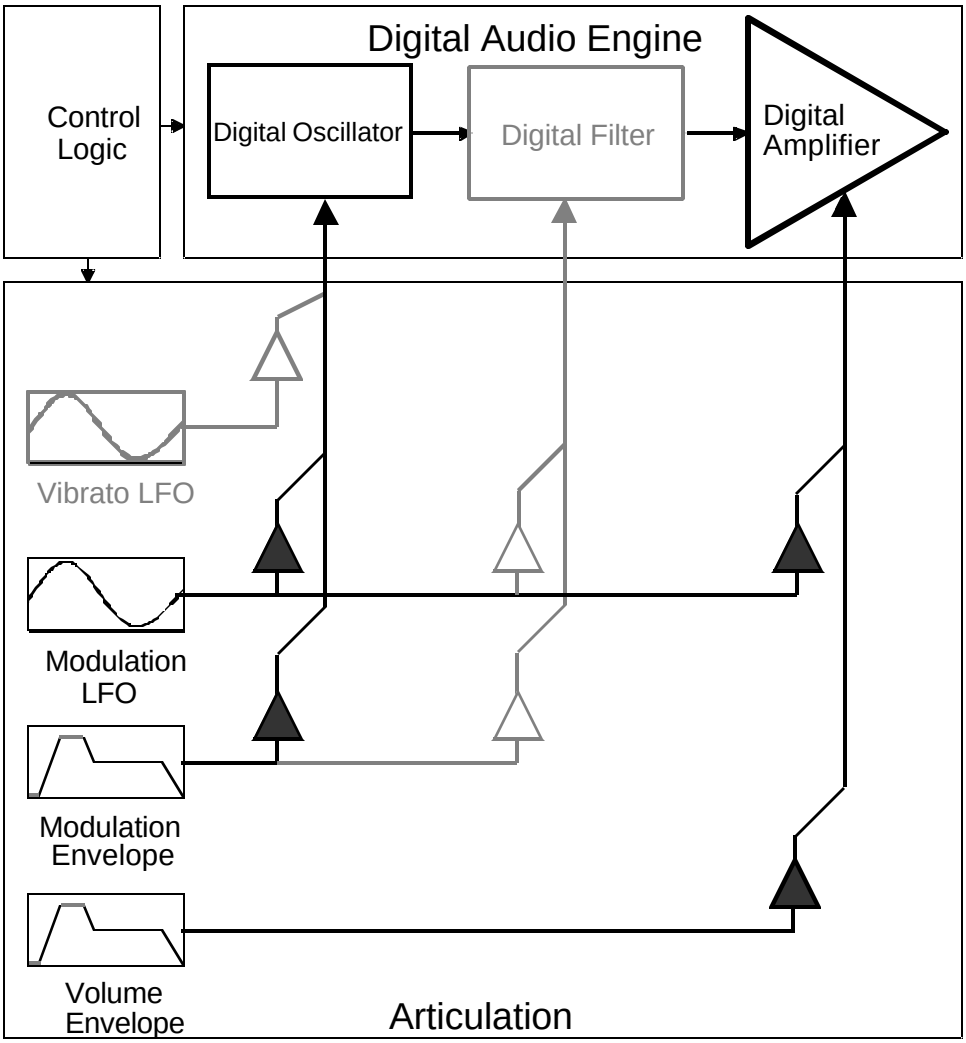


Figure 1 — Block Diagram of Mobile DLS System Architecture

The mobile DLS synthesizer depicted in Figure 1 contains the "common part", which is always implemented in a Mobile DLS synthesizer, as well as the optional group of processing blocks mentioned above. The optional blocks are the Loop and Release loop type (not shown), two 6-segment envelope generators replacing the 4-segment envelope generators, Digital Filter (DCF), and Vibrato LFO.

A synthesizer supporting only the common part should not render Mobile DLS content that uses the optional group of processing blocks. If a manufacturer decides to play incompatible Mobile DLS content, the device should inform the user about the incompatible content. The playback of incompatible Mobile DLS content should be avoided to ensure predictable playback behavior across different Mobile DLS synthesizers.

1.2.4.3.1 Signaling Mechanism for Mobile DLS Features

The Mobile DLS system supports an external signaling mechanism for indicating optional Mobile DLS synthesizer configurations. The purpose of the external signaling is to support consistent playback behavior and efficient rendering of Mobile DLS instruments. The optional Mobile DLS features can be grouped into a single control group to simplify the signaling. In this way, a common control signal can be used for the entire control group, and no separate methods are needed to control each unit inside the group.

External signaling can, for example, be implemented via the DLS parser. One signaling means for detecting whether the optional processing blocks are used is to check if WLOOP_TYPE_RELEASE is used or if Vol EG Delay Time, Vol EG Hold Time, Mod EG Delay Time, Mod EG Hold Time, Initial Fc, or Vib LFO to Pitch overrides the

corresponding default value as given in Table 6, section 1.13.1. The values are 0x7FFFFFFF for Initial Fc and 0 for Vol EG Delay Time, Vol EG Hold Time, Mod EG Delay Time, Mod EG Hold Time, and Vib LFO to Pitch. Other methods can also be used to check whether the content uses the optional processing blocks.

1.3 Minimum Device Requirements

Device requirements are essential for the applicability of the DLS technology for mobile applications. Table 1 compares the device requirements for Mobile DLS to the existing DLS 2.1 and DLS 1.1 specifications. This specification does not review the DLS-1 system or the DLS-2 system after this section. Instead, the specification defines how to support Mobile DLS instruments. A device must meet the minimum requirements shown in Table 1 to be considered compliant with the Mobile DLS specification.

Minimum Device requirements	**Mobile DLS**	**DLS-2.1 system**	**DLS-1.1 system**
Minimum number of simultaneous voices.	5 voices	32 voices	24 voices
Minimum playback sample rate	8 kHz	22.05 kHz	22.05 kHz
Total uncompressed Mobile DLS instrument content (including waveform and articulation data)	7 kbytes, 15 kbytes (optional, required for devices supporting the optional group of processing blocks)	Undefined	Undefined
Synthesizer voice architecture units: Digital filter (DCF) and Vibration LFO	Optional (recommended)	Mandatory	Not supported
Minimum number of articulation connections for instruments	8192 (Not exceeding the total uncompressed Mobile DLS content memory limit)	8192	128 sets of articulation data
Minimum number of regions for instruments	1024 (Not exceeding the total uncompressed Mobile DLS content memory limit)	1024	128 regions
Note: These figures are minimum requirements only, and should not be viewed as limits to device capabilities. To encourage product differentiation and future capabilities, manufacturers are encouraged to exceed these minimums and document their capabilities so that content authors and consumers may know what those capabilities are.			

Table 1 — Comparison of DLS Model Minimum Requirements

1.4 Synthesizer Design Detail

1.4.1 Control Logic

The control logic implementation is relatively simple, a shown in Figure 2. The control logic receives MIDI Bank Select and Program Change commands to select the instrument on a particular channel, and MIDI Note-On and Note-Off events to play notes on the same channel. The combination of instrument choice (bank select and program change) and note is used to select a specific configuration of the articulation modules and digital audio engine to perform the note. The control logic must select a specific sample for the digital audio engine to play. The sample is indirectly accessed through a region. The region defines the key range and velocity range used by the control logic to select the sample. It also determines a preset value for the overall amplitude of the sample and a preset tuning. The sample, in turn, defines the actual chunk of digitized sound, along with control parameters such as loop points, sample rate, and so on.

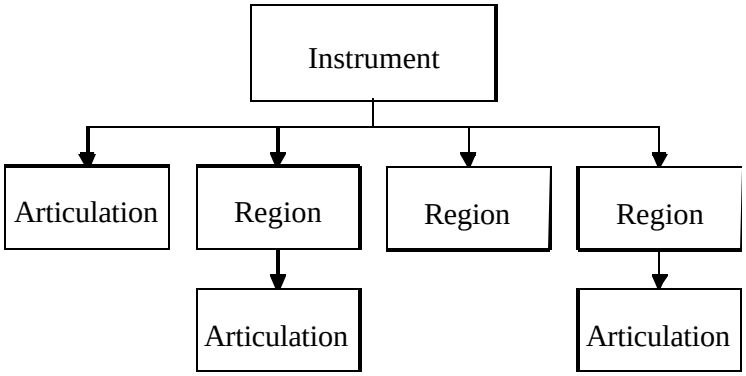


Figure 2 — Control Logic

1.4.2 Articulation Data

[TW2]The *articulation* is a complete configuration of articulation modules and their connections, including the envelope generators and LFOs. These define how the note should be articulated as it plays.

An instrument is composed of one or more regions, which in turn define the range of notes and velocities, which that particular region should play, the sample that should be played, and the articulation that should be applied to the sample data by the digital audio engine. Layering is facilitated through the use of overlapping note regions, with each region using up an oscillator in the device. Velocity splits are facilitated through the use of overlapping note regions with different velocity ranges, while key splits are facilitated similarly by overlapping velocity ranges with different note ranges.

Articulation data can either be global to the entire instrument, or it can be local to a specific region within the instrument. Local articulation data always takes precedence over global articulation data. Global articulation data determines the default behavior of the instrument. Local articulation can be used to modify the behavior of a particular region. Mobile DLS content creators should prefer global articulation whenever feasible for memory efficiency.

A connection in the articulation data consists of a Destination within the synthesizer, a Source, and a Control, either of which may be internal or external. The combination of Destination, Source, and Control make a unique connection. The Mobile DLS specification defines default behavior or "implied" connections in the absence of any articulation data. The synthesizer executes each implied connection unless the articulation data in the instrument specifically overrides it.

For example, the default connection "Mod LFO Frequency", which has Source "SRC_NONE", Control "SRC_NONE" and Destination "DST_LFO_FREQ" (See Modulation Routing, Table 5) sets the Modulation LFO frequency to 5 Hz. To override this, the articulation must specify the exact same Source, Control and Destination along with the new frequency.

Global articulation data either augments, or as shown above, overrides the implied connections, creating a new set of default connections for the instrument. Similarly, local articulation data augments or overrides the implied connections, and overrides any global articulation data.

This document describes a number of sources, destinations and various transforms that can be applied to the connections between them. It also describes a fixed set of routing connections that *must* be implemented in order for a synthesizer to be Mobile DLS compliant. However, these fixed connections represent a small fraction of the universe of connections that are possible. Developers are free to use these other connections in a Mobile DLS file, provided that they are protected by Conditional Chunks to guard them from unwitting access by devices that are not expecting such connections.

1.4.3 Conditional Chunks

Mobile DLS supports both DLS-1 and DLS-2 chunks, which can be implemented by supporting the DLS-2 file format. In DLS-2 files, DLS-1 chunks and DLS-2 chunks can be supported using conditional chunks. DLS-2 introduced a new functionality into the control logic, namely a block called a "Conditional Chunk." Conditional Chunks can be used to create libraries that are both forward and backward compatible. As an example, an instrument in a library may contain DLS-1 chunks, DLS-2 chunks, and chunks for a proprietary device. The file parser will then select the appropriate chunks for the specific device in use. A detailed description of the Conditional Chunk can be found in Section 2, "RIFF Structure (page 37).

1.4.4 Note Exclusivity

A Mobile DLS Device has provisions for two forms of note exclusivity on a MIDI channel basis, shown in Table 2. The first form of exclusivity involves notes on a MIDI channel with the same MIDI note number. DLS-2 introduced the concept of a shutting down an oscillator, to give the sound developer more control over polyphony without degrading sound quality. When a note exclusive event occurs, the Volume envelope proceeds directly to the release phase, but using the time constant EG1_SHUTDOWNTIME, instead of EG1_RELEASETIME.

By default, if a MIDI Note-On event is received and there are oscillators previously assigned to the same MIDI note and MIDI channel that have not received a Note-Off event (or Note-On event with a velocity of 0), the control logic will immediately shutdown those oscillators. However, there is a *Non-Self-Exclusive* flag in the *fusOptions* field in the region header chunk that, when set, will defeat this logic. When the *Non-Self-Exclusive* flag is set, Note-On events for a particular MIDI note number on a MIDI channel do not cause a shutdown of oscillators assigned to the same MIDI note and MIDI channel. When the *Non-Self-Exclusive* flag is not set and the synthesis engine receives a second Note-On event of the same MIDI note number on the same MIDI channel, the second Note-On will cause a shutdown of the first note. The *Non-Self-Exclusive* flag is off by default.

The second form of note exclusivity is useful for drums and sound effects. Each region can be assigned *a Key Group*. If a Note-On event is received and there are oscillators that have been assigned to play a region that has the same *Key Group* number as the region for the new Note-On, those oscillators are shutdown. As an example, this can be used to create mutually exclusive Open, Closed and Pedal High Hat sounds for a drum group. A second example might be in a sound effects collection to stop a squeaking door sound when the sound of the door closing is triggered.

Description	**Mobile DLS instrument data**
Oscillator Shutdown with EG1_SHUTDOWNTIME	Yes
Channels capable for mutually exclusive mode	Any channel

Table 2 — Supported Note-Exclusivity Functionality

1.4.5 Voice Allocation

Voice Allocation is the means by which digitally controlled oscillators are allocated to play samples as dictated by the instrument parameters and the MIDI data stream. Ideally, there should be enough oscillators to play every note and sound effect in the score. However, this may not always be the case. Some devices may have more oscillators than others, and the composer may create a score that requires a higher polyphony playback than the device can support. If the MIDI data stream calls for more oscillators than the device is capable of producing, the composer must be able to accurately predict the result.

For SP-MIDI content the allocation of playback resources is determined in part by scalable playback of those MIDI Channels that do not exceed playback capabilities of the device. SP-MIDI defines a dynamic (content dependent) Channel Priority order that overrides any other channel prioritization scheme. A more detailed description of SP-MIDI voice allocation appears in [6].

The Mobile DLS synthesizer must default to a specific channel priority order: 10 (highest), 1-9, 11-16 (16 lowest). The synthesizer must respond to SP-MIDI MIP messages, i.e. by setting the channel priority order according to the

received MIP message. Choice of an oscillator allocation (note stealing) algorithm is left to the discretion of the implementer, so long as the channel priorities are handled in the manner just described.

Mobile DLS devices do not have to support DLS StaticVoice Allocation sub-ID #2 with parameters 03 and 04. The DLS System Exclusive message cannot be used to turn off SP-MIDI based voice allocation.

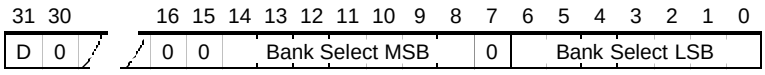
Description	**Mobile DLS instrument data**
DLS Static Voice Allocation	No
Default channel priority order	10, 1-9, 11-16
SP-MIDI channel priority order always applied to SP-MIDI content	Yes
Note stealing algorithm	Manufacturer dependent

Table 3 — Supported Voice-Allocation Functionality

1.4.6 Bank Select and Program Change

The Mobile DLS device must support MIDI Bank Select and Program Change messages as the method of selecting the instrument to be played on a MIDI channel. The Bank Select address space consists of 16,384 banks, represented by the MIDI Controller Change MSB and LSB Messages (controllers 0 and 32, respectively), with each bank supporting up to 128 instruments, for a total address space of over 2 million instruments.

The device does not take any direct action upon receipt of Bank Select messages, but saves the state for future reference. Upon receipt of a Program Change message, the device should refer back to the previously received Bank Select values to decode the correct instrument to be played. The Bank Select LSB and MSB are concatenated as 8-bit bytes to form *ulBank* as shown:



Content authors must send Bank Select MSB, Bank Select LSB, and Program Change messages in that order to insure that the correct instrument is selected. Bit 31 is informative and indicates a drum instrument. A MIDI channel that references *ulBank* with MSB 0x79, via a Bank Select and Program Change, shall be treated as a melodic channel using the default melodic bank on the device. A MIDI channel that references *ulBank* with MSB 0x78, via a Bank Select and Program Change, shall be treated as a drum channel using the default drum bank on the device.

The Mobile DLS device shall provide a dedicated location for a General MIDI or General MIDI 2 instrument bank. The minimum requirements for the General MIDI sound bank follow the SP-MIDI specification requirements [6]. The GM or GM2 percussion programs shall appear at bank select MSB=0x78 LSB=0x00, Program 0x00. If a GM instrument set (as specified in [4]) is used, the melodic programs shall appear at bank select MSB=0x79 LSB=0x00. If a GM2 instrument set is used, the melodic programs shall appear at bank select MSB=0x79, LSB=0x00 through 0x09, as specified in [5].

Any custom program(s) from Mobile DLS content files may use the same bank and program number(s) as any of the Mobile DLS device's built-in GM/GM2 programs; if so, they will temporarily override the corresponding built-in programs until they are unloaded.

1.4.7 Phase-Locked Samples

A Mobile DLS device does not need to support Phase-Locked Samples.

1.5 Digital Audio Engine

The digital audio engine consists of a Digital Oscillator, Digitally Controlled Filter, and Digitally Controlled Amplifier.

1.5.1 Digital Oscillator

The Digital Oscillator takes as its input an audio sample data and plays it back at a rate controlled by the articulation data. It must perform interpolation on the sample data in order to convert it from the sample rate at which it was recorded to the output sample rate, applying any pitch translation required by the articulation data.

The sample defines whether it is to be played once through or looped. If looped, the sample supports one looped region, as defined by a pair of fixed sample data point indices. The loop start specifies the first sample data point in the loop, while the loop length specifies the number of samples in the loop. There is no fractional component to the loop points.

Mobile DLS supports the Forward Loop type shown in Figure 3 and the Loop and Release type shown in Figure 4.

In the Forward Loop, the device begins playback at the start of the sample. Upon reaching the end of the loop as defined by the loop length, the device returns to the start of the loop. Playback continues in this fashion until the end of the release segment of the envelope.

In the Loop and Release loop, the device begins playback as in the simple loop form. However, at the time that the MIDI Note-off event is received, the device does not return to the beginning of the loop, instead playing straight to the end of the sample. It is very important that the device not jump directly to the release segment, but continue playing from wherever it may be in the loop straight through to the end of the sample, thus maintaining continuity in the samples.

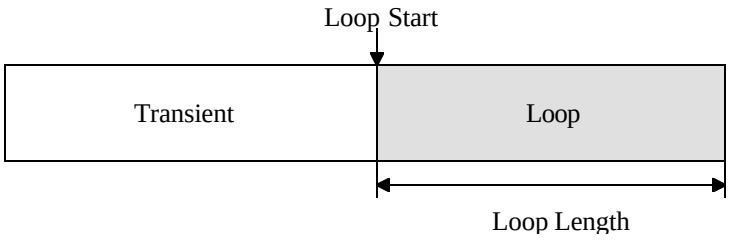


Figure 3 — Forward Loop

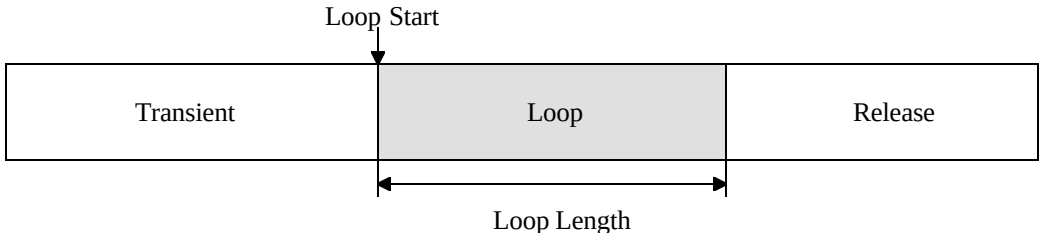


Figure 4 — Loop And Release

The sample structure defines the MIDI note number, fine tune, and sample rate of the recorded sample, providing all the information required for the digital oscillator to pitch shift the sample appropriately for the intended pitch. The Digital Oscillator must be capable of pitch shifting the sample with a range of up two octaves and down four octaves. This pitch shift is the sum total of all tuning parameters, including sample fine tune and MIDI pitch bend.

Description	Mobile DLS instrument data
Loop type	Forward Loop and Loop and Release
Waveform data format	RIFF WAVE PCM format

Table 4 — Supported Digital Oscillator Functionality

1.5.1.1 PCM Sample Data Format

When the wFormatTag is equal to WAVE_FORMAT_PCM, the sample data can be 16-bit two's complement or 8-bit offset two's complement (often referred to as 8-bit unsigned data, with zero represented as 0x80). This complies with the both the RIFF WAVE PCM format and common practice on the Macintosh. In order to use the 8-bit format, it is usually necessary to convert it to standard two's complement. To convert between offset two's and standard two's complement (signed 8 bit, -128 to +127), XOR each byte of the source sample value with 0x80.

1.5.2 Digitally Controlled Filter

The input to the Digitally Controlled Filter is the audio data stream produced by the Digital Oscillator. The input samples contained in the audio data stream are then filtered as required by the articulation data. The filter is defined to be a 12 *dB*/octave cutoff slope lowpass digital filter with a transfer function of the form:

$$H(z) = \frac{K}{1 + b_1 z^{-1} + b_2 z^{-2}}$$

This transfer function represents a single complex pole pair located within the unit circle at radius r and angle θ where:

$b_1 = -2\, r \cos\theta$

$b_2 = r^2$

$K = g\,(1 + b_1 + b_2)$

where g is a gain factor which depends on the intended resonance. This is done in order to keep the perceived loudness approximately constant. Otherwise, the addition of resonance will make the sound louder. As the resonance is increased, g is decreased as follows:

$$g = 10^{-resonance/40}$$

where the resonance is expressed in decibels. Another way to look at this is that the DC gain of the filter is attenuated by a factor equal to half the decibels of the resonance value.

When g is expressed in decibels, the relationship between g and resonance is:

$$g_{dB} = \frac{-resonance}{2}$$

For example, if the specified resonance is 20 *dB*, g_{dB} is –10 *dB*. Since g is multiplicative to the numerator of the transfer function, it is equivalent to apply it after the filter by summing g with the other gain sources for the signal. However, it is desirable to perform this within the filter when bit-width becomes a limiting factor to headroom within the filter delay memory.

Such filters can either express a non-resonant low-pass filter with a frequency response similar to Figure 5, or a filter with a marked resonant peak with a response similar to Figure 6.

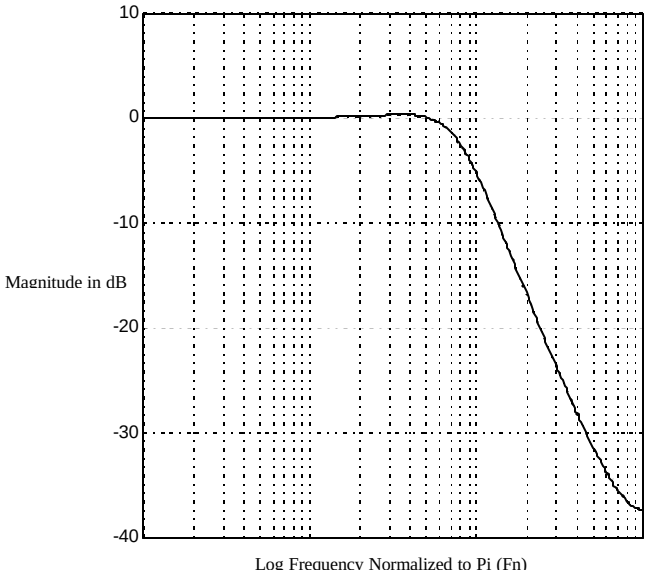


Figure 5 — Non-Resonant Low-Pass Filter

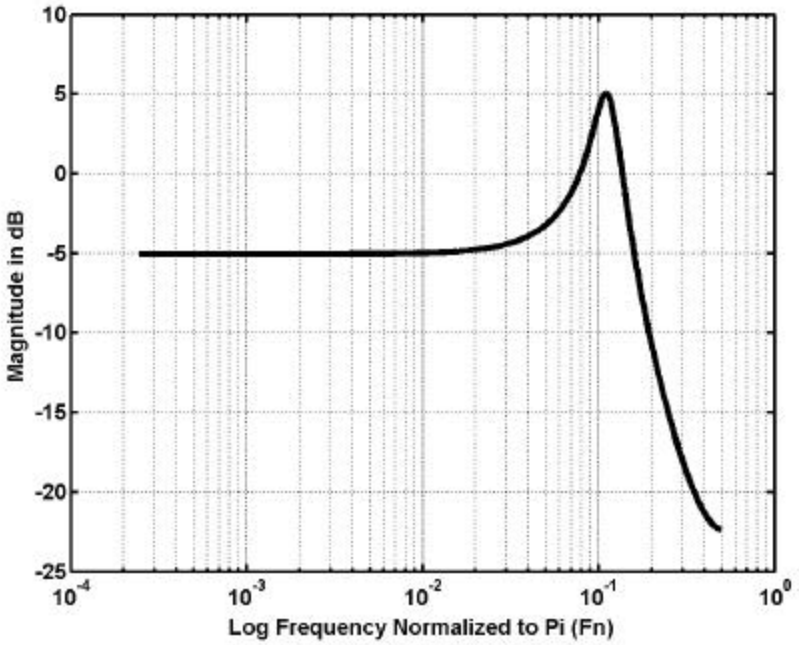


Figure 6 — Resonant Filter

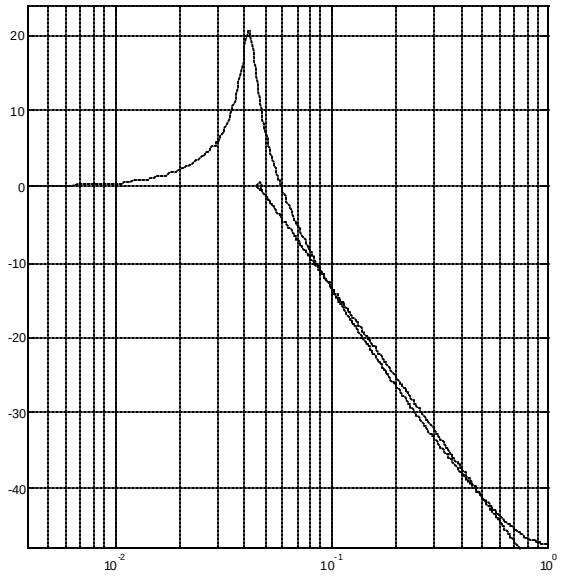


Figure 7 — Filter Cutoff Frequency

The filter will be characterized by a **resonance** specified in 32-bit gain units (*dB*/655360), which will represent the ratio of the gain at the top of the resonant peak to the gain at DC. A filter with a specified resonance of 0 will not have any resonant peak. It is illegal to specify a negative gain value for resonance. The default resonance value is 0.

The filter will be characterized by a cutoff frequency in 32-bit absolute pitch units. This frequency is the frequency of the point of intersection of the DC gain line of the filter (a horizontal line on the frequency response) with the asymptotic cutoff line of the filter (a downward sloping line at 12 *dB*/octave). Because of the deviation from asymptotic behavior near the Nyquist frequency, this asymptotic cutoff line will be the line drawn at –12 *dB*/octave passing through the point one octave below the Nyquist frequency on the filter frequency response.

This method is used rather than the standard –3 *dB* point because this makes the filter cutoff frequency less sensitive to the height of the resonant peak. Note that in most cases, when calculated in this manner, the difference between the cutoff frequency and the center frequency of the resonant peak (when resonance is present) is very small. A generalized line in gain-frequency space at –12 *dB*/octave is defined by the following equation:

$$h_{-12dB} = Af^{-2}$$

In order to find the line that intersects the filter response curve at the Nyquist/2, one must solve for A:

$$A = \frac{F_s{}^2 h_{filter}(F_s/4)}{16}$$

where $F_s \equiv$ sample frequency and $h_{filter}(F_s/4)$ is the gain of the filter at the Nyquist/2. The filter cutoff frequency is the intersection of that line and the DC gain of the filter:

$$F_c = \frac{F_s}{4}\left(\frac{\sqrt{1-2r\cos q + r^2}}{\sqrt[4]{1+2r^2\cos(2q)+r^4}}\right)$$

So, q can thus be expressed as a function of F_c and r:

$$q = \arccos\left(\frac{-F_s^4\left(r^2+1\right)+16F_c^2\sqrt{2F_s^4\left(r^4+1\right)-256F_c^4\left(r^2-1\right)^2}}{2r\left(256F_c^4-F_s^4\right)}\right)$$

The resonance of the filter is given by the following expression:

$$resonance = 20\log_{10}\left(\frac{1-2r\cos q+r^2}{(1-r^2)\sin q}\right)$$

where r and q are the radius and angle of the poles of the filter. Note that this is decibels, not absolute gain units. Given this relationship, the radius can be calculated as a function of resonance and θ as shown in the following equation:

$$r = \frac{\cos q + \sin q\sqrt{10^{resonance/10}-1}}{10^{resonance/20}\sin q+1}$$

where resonance is specified in decibels. The derivation of these equations is left as an algebraic exercise for the reader.

1.5.3 Digitally Controlled Amplifier

The final module in the signal path is the digitally controlled amplifier (DCA). It modulates the amplitude of the digital oscillator signal, responding to controls from the articulation section, of which the amplitude envelope is probably the most important.

For the purposes of this document, the maximum output of the DCA is assumed to be 0 *dB* or unity gain, though as a practical matter, some attenuation may be required prior to summing all the oscillator outputs together to form the final synthesizer output. Therefore, all changes in the DCA are measured in *dB* units of gain from the 0 *dB* reference point, and all units of *dB* in this document refer to voltage, rather than power. Positive units of gain represent an increase in output voltage, and negative units represent a decrease in output voltage. The MIDI pan signal is fed through a transform that converts to left and right amplitude signals. These signals are fed separately into the DCA to generate stereo output from the monophonic signal entering the DCA.

The gain of the DCA can be varied dynamically through the use of modulators. Since the steps in the input sources can be rather severe, the designer must minimize audible artifacts (such as "zipper" noise) by filtering the control source. This can be accomplished by interpolating between control events (such as MIDI controller sources).

1.6 Articulation Modules and Connections

The Articulation modules include two low frequency oscillators (LFOs), two envelope generators, and a number of MIDI controller inputs. A configuration is defined by connecting this base set of modules and setting the controls for each one.

1.6.1 Signal Flow

The Control Logic module translates MIDI note and sustain events into region/sample choice and associated control signals, including Key (the note number), Velocity (note emphasis), and Gate. Gate defines the start and end times of

the note, and is used to trigger modulators, such as the LFO and envelope generators. Gate is also used to control the loop status in the digital oscillator for Loop and Release samples.

The digital oscillator unit plays the sample. It is primarily controlled by the Pitch Sum node, which accumulates signals from modulation sources to deliver the final pitch for the instrument.

The output samples of the digital oscillator flow into the digital filter. The cutoff frequency and the resonance characterize the operation of the filter.

The digitally controlled amplifier takes the output samples of the digital filter. It is controlled by two signals: the Volume Sum node, which accumulates signals from modulation sources to deliver the final volume of the instrument; and the MIDI pan signal, which sets the stereo position of the stereo digital output.

Control signals from MIDI modulation sources, such as Volume and Pitch Bend, flow through connections which use their gain values and transforms to define the range of the control. The Volume Sum node (DST_GAIN) can be viewed as a summing node if signals are represented in *dB*, or as a multiplier if signals are represented in linear units. 0 *dB* is defined as the maximum output level at the Volume summing node (DST_GAIN).

1.6.2 Connection Block

Each connection can be viewed as a block that defines the *usSource*, *lScale*, *usControl*, optional *usTransform*, and *usDestination*.

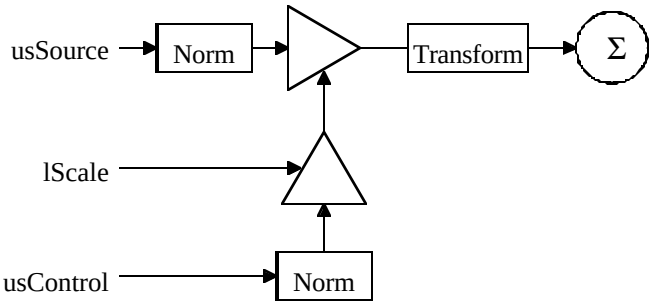


Figure 8 — Mobile DLS Connection Block

Each external source goes through a normalization block that scales and applies certain types of transforms to the signal. Further transforms may be applied at a later stage. The normalization block scales the incoming values to a range of -1 to +1 for a bipolar transfer function, or 0 to 1 for unipolar transforms. The output of the transform may also be inverted at this stage.

All connections to a single destination are summed together to produce the final input for that destination. The pseudo-code for a connection block is as follows:

```
        usDestination = usDestination + usTransform(Norm(usSource) * (Norm(usControl) *
lScale))
```

Controlled Scaled Source connection blocks use the *usControl* input to modulate or control the scaled *usSource* input routed to the *usDestination*. A Controlled Scaled Source connection consists of *usDestination*, *lScale*, *usSource*, *usControl* and an optional *usTransform*. The result is a signal that is modulated by both the *usSource* input and the *usControl* input. An example of this is the LFO Mod Wheel to Pitch connection block, which allows the Modulation controller (CC1) to determine the amount of LFO output applied to *usDestination* DST_PITCH, which is the frequency of the digital oscillator.

Any unconnected inputs are seen as a full scale signal, so that the output of the normalization block is either +1 (for non-inverted signals) or -1 (for inverted signals). With no input connected to *usControl*, the connection block takes on the simpler form of a Scaled Source, where *lScale* sets the amount of modulation that flows from the source input to the destination.

Scaled Source connection blocks scale the *usSource* input routed to *usDestination*. A Source Scale connection block consists of *usDestination*, *lScale*, *usSource*, and an optional *usTransform*. The *usControl* input is set to SRC_NONE. The result is a signal that is modulated by the *usSource* input and applied to the *usDestination*. An example of this is

the EG2 to Pitch connection block, the amount of modulation applied to pitch from the Pitch Envelope to *usDestination* DST_PITCH.

With no input on either *usSource* or *usControl*, the connection block takes on the simplest form of a Scaled Connection, which is a bias applied to the destination. As examples, Scaled Connections are used to alter the playback frequency, to set the initial gain of the digital amplifier, or to set the initial cutoff frequency or resonance of the filter.

1.6.3 Default, Global, and Local Articulation Data

The Mobile DLS specification defines default connections, which determine the behavior of the instrument in the absence of specific articulation data. An important concept of DLS is that each combination of *usSource*, *usControl* and *usDestination* defines a unique connection for that region. In the absence of articulation data for a specific connection, the default values for that connection will be used. If the articulation data contains such a connection, it takes precedence over the default values *for that connection only*. Duplicate connections within a single region are not allowed, and the behavior of the device is undefined under those circumstances. *Note that Mobile DLS devices do not give any preference to <art2-ck> chunks over <art1-ck> chunks, they are treated identically*.

For example, if a region contains an <art1-ck> chunk and an <art2-ck> chunk, and the EG1 Release Time is set to different values in the two articulation chunks, the actual release time is indeterminate for Mobile DLS devices. To avoid this condition, the <art1-ck> should be guarded by a <cdl-ck> conditional chunk that excludes devices that support Mobile DLS. This conditional chunk will cause the Mobile DLS device to ignore the <art1-ck> and read only the <art2-ck>.

Global articulation data defines the behavior of any regions that do not have applicable local articulation data, either because the local articulation data does not exist or the device fails to meet the conditional criteria. In the event that no applicable local articulation data exists, the instrument behavior for that region is defined by the global articulation data. If local articulation data does exist that meets the conditional criteria of the device, the local data will override all global articulation data. In other words, global articulation data, and its implied default connections, must be treated as a monolithic block of data: used as a whole; or discarded in favor of local articulation data. This is in contrast to default articulation data, which is overridden on a connection by connection basis, both at the global and local level. This is an important distinction to understand and forms the basis for the parsing code.

1.6.4 Controller Theory

All control signals in the DLS synthesizer model are normalized to a range from -1 to 1. They may also go through an optional series of input transforms. Internal sources are assumed to generate their outputs already normalized.

The following equations all assume that the controller input signal "*Input*" and the controller range "*Range*" are non-negative integers. They also assume that the controller input minimum value is zero and the controller input maximum value is "*MaxValue*". If necessary, the following saturation operation is recommended in order to ensure the upper bound of the controller input signal:

```
if (input > MaxValue)
    input = MaxValue;
```

For the linear or switch input transforms *"MaxValue"* is equal to "*Range – 1"*, whereas for the concave or convex input transforms the controller input signal is saturated at a *"MaxValue"* equal to *"Range*(127/128)"*. This difference in *"MaxValue"* for the concave and convex transforms ensures that their output values are equal whether a 7-bit controller is used, or a 14-bit controller with 7 least significant bits equal to zero is used.

For example, MIDI controllers are assumed to have input values from 0 to 127, for a range of 128. On the other hand, the pitch bend controller, whose range is 16,384, is assumed to have input values from 0 to 16,383 for the linear or switch transforms, and from 0 to 16,256 for the concave or convex input transforms. For example, the formula for normalizing the Modulation controller (CC1) for linear, unipolar output is as follows:

```
Output = CC1 / 128
```

Another example, normalizing the Pitch Wheel for bipolar linear output is as follows:

```
Output = 2 * (PitchWheel / 16384) - 1
```

1.6.5 Transforms

In many cases, the input data to a module must be transformed into units that can be accepted by that module. For example, the linear output of an envelope generator needs to be converted into a format compatible with the DCA, typically an exponential transform. Likewise, the link between a MIDI volume input and the DCA must undergo a similar transform. To accomplish this, the connection mechanism allows for an optional transform to be defined for every connection. Only the transforms specified in the Default Routing Modulation Table 5 are required for Mobile DLS devices, thus all the specified connections are fixed. However, different routing is allowed within articulation list chunks that are marked with proprietary conditional chunks to allow manufacturers to support specific features in their own Mobile DLS devices.

1.6.5.1 Invert

The Invert flag inverts the polarity of the controller signal when set. For MIDI controllers, this operation must be performed before the controller input is normalized, in order to reflect the asymmetry of the MIDI controllers. The inversion algorithm is defined as follows

```
if (invertFlag)
       input = MaxValue – input;
```

1.6.5.2 Bipolar

The Bipolar flag maps the controller input signal to a range of -1 to +1. For non-linear input transforms, this introduces a slight complication, as the non-linearity must be mirrored in both the positive and negative quadrants. Thus, for the concave and convex input transforms the following algorithm can be used to perform the bipolar mapping:

```
if (bipolarFlag)
{
       value = (2 * input) - MaxValue;
       output = sgn(value) * InputTransform(abs(value));
}
else
       output = InputTransform(input);
```

On the other hand, for linear and switch input transforms, the algorithm is:

```
if (bipolarFlag)
{
       value = InputTransform(input);
       output = (2 * value) - 1;
}
else
       output = InputTransform(input);
```

1.6.5.3 Input Transforms

The Input Transform maps all controller signals to a range of 0 to 1. Supported input transforms are shown in Figure 9.

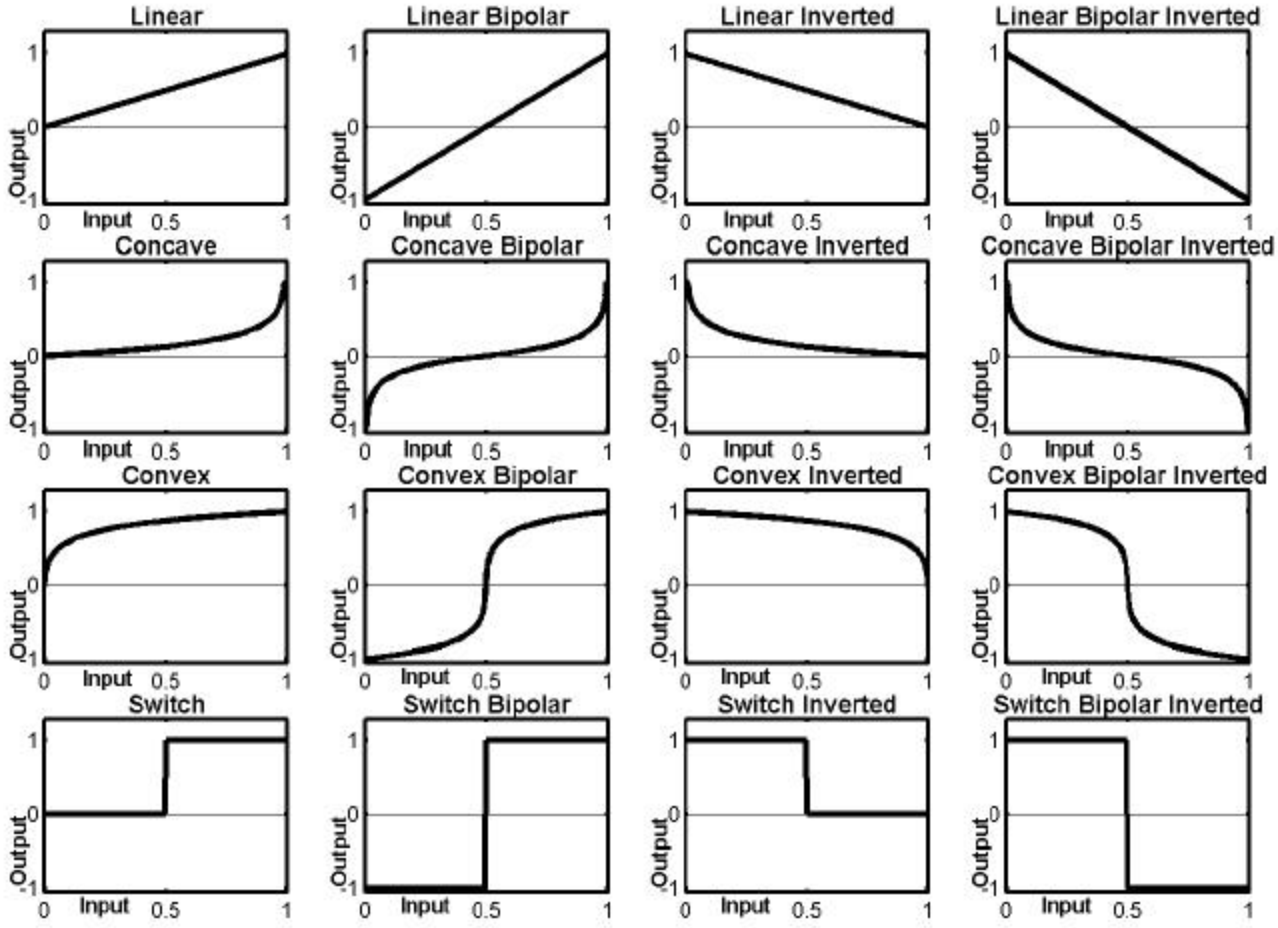


Figure 9 — Transforms

The various transforms and their associated equations are:

Linear: Input values map linearly as defined by the following equation:

Output = Input / Range

Concave: Input values are mapped in a concave fashion as defined by the following equation:

```
For Input > (1.0 – 10^-12/5)*MaxValue,
      Output = 1.0

For Input = (1.0 – 10^-12/5)*MaxValue,
      Output = –(5/12)*log10(1.0 – Input/ MaxValue)
```

Convex: Input values are mapped in a convex fashion as defined by the following equation:

```
For Input < (10^-12/5)*MaxValue,
      Output = 0.0

For Input >= (10^-12/5)*MaxValue,
      Output = 1.0 + (5/12)*log10(Input/MaxValue)
```

Switch: Input values are mapped to either zero or one as follows:

```
For Input values < Range / 2, Output = 0
For Input values = Range / 2, Output = 1
```

1.6.5.4 History of the Concave Transform

The Concave Transform was created to maintain compatibility with existing General MIDI implementations. Its name comes from the shape of the curve when graphed as a linear volume output. In DLS-1, it was defined as a series of fixed connections from MIDI controllers 7 and 11, and velocity, to attenuation in dB using the following formula:

$$atten_{dB} = 20 \times \log10(127^2 / Input^2)$$

To map this fixed transform onto the flexible routing model found in DLS-2, it is necessary to adjust the output of the transform so that it produces the desired curve in the range 0 to 1, such that when scaled at 96 dB and applied to the DST_GAIN summing node, it produces the same output curve as that of the DLS-1 and GM devices.

Due to these changes, in this Mobile DLS specification the convex transform corresponds to the theoretical definition of the concave function and the concave transform corresponds to the theoretical definition of the convex function.

In DLS2.1, the scaling factor of 5/12 was derived from the 96 dB scaling factor, as shown here:

The attenuation formula from DLS1.0:

$$20*\log10((127/Input)^2)$$

is equal to:

$$40*\log10(127/Input)$$

Dividing by 96 to scale back to a range of 0–1:

$$(40/96)*\log10(127/Input)$$

reduces to the equivalent:

$$(5/12)*\log10(127/Input)$$

Since this was attenuation, whereas DLS-2 uses gain, we invert the input, so it becomes:

$$(5/12)*\log10(127/(MaxValue - Input))$$

The lScale values for connections to DST_GAIN are intended to be negative numbers. For example, the default lScale value for velocity to gain is –96dB. And the input transform is defined to be inverted concave. So the default velocity to gain calculation is:

$$gain = -96 * (5/12)*\log10(127/(127 - (127 - velocity)))$$

Selecting a velocity of 127 produces 0dB of gain:

$$gain = -96*(5/12)*\log10(127/(127 - (127 - 127))) = 0dB$$

Selecting a velocity of 0 produces -96dB of gain – it forces the inverted concave transform to equal 1.0 because the input is equal to MaxValue:

$$gain = -96*(5/12)*\log10(127/(127 - (127 - 0)))$$

$$gain = -96*1.0 = -96dB$$

It should be noted that 96dB is an approximation of 16-bit dynamic range. The true value is 20*log10(65536), or 96.3296dB. However, the approximation cancels out when re-linearizing the final output to gain, only producing a difference when the output is saturated to 1.0.

The concave transform in DLS2.1 has some problems. First, it was defined using Range, whereas the intended implementation used MaxValue. Second, the output is greater than 1.0 before the input reaches MaxValue when applied to a 14-bit controller. Third, when the 7 least significant bits of a 14-bit controller are zero, the output values do not equal the output values using a 7-bit controller.

The first and third problems are addressed by using (127/128)*Range instead of Range. The second problem is addressed by the modified saturation threshold equation.

1.6.5.5 Output Transforms

There are no output transforms defined in Mobile DLS. The bits in the output transform field must all be set to zero for compatibility with future versions of DLS.

1.7 Generators

Each oscillator has two low frequency oscillator generators and two envelope generators. Each generator has a set of controls to determine its behavior.

1.7.1 Low Frequency Oscillator

The LFO generates a periodic waveform that is used to modulate the pitch or volume of the instrument. The LFO waveform may be either a triangle or sine wave.

1.7.1.1 Frequency

The frequency control sets the wave frequency between 0.1 Hz and 20 Hz in absolute Pitch Cents.

1.7.1.2 Start Delay Time

The LFO starts after a preset delay time from 10 milliseconds to 10 seconds in Time Cents. The device is not allowed to produce any artifacts when the LFO starts after a delay. If the device cannot synchronize the LFO phase to the end of the delay cycle, it is recommended that the LFO output be ramped up to full scale over a one cycle period to eliminate the possibility of introducing a DC offset into the output path.

1.7.2 Envelope Generator

The envelope generator creates a six-segment envelope. The segments correspond to the attack, decay, sustain, and release phases of the traditional analog ADSR (Attack, Decay, Sustain, Release) design, plus two additional phases: Delay and Hold. The diagram in Figure 10 shows the relationship between the various envelope phases and the output levels.

Upon receiving a Note-On event, the envelope enters the Delay phase, and the output of the envelope generator remains at zero throughout this phase. At the completion of the Delay phase, the envelope generator begins the Attack phase, where it transitions from zero to full scale in linear fashion. After the Attack phase, the generator enters the Hold phase, where the output holds at full scale for the specified time. After the Hold phase, the generator enters the Decay phase, where it falls to the Sustain Level in an exponential decay fashion.

Upon reaching the Sustain Level, the generator remains at this level until a Note-Off event (or a Note-On event with a velocity of 0) is received. Upon receiving a Note-Off event, and regardless of the previous state, the generator enters the Release phase, with the output falling in an exponential fashion from its current level until it reaches the pre-determined zero level again (typically -96 *dB*).

Decay and Release Time Constants are actually rates defined as the time for the signal to decay from full scale to -96 *dB*, at which point it is assumed that most implementations will make an abrupt transition to infinite attenuation.

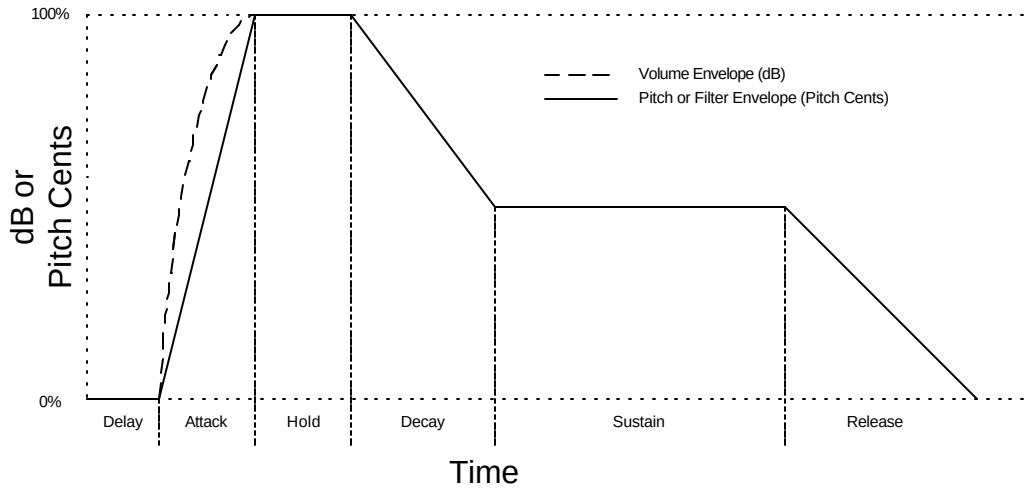
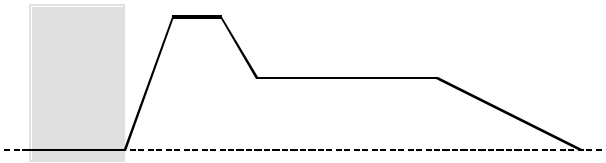


Figure 10 — Envelope

IMPORTANT: The following envelope diagrams show all envelope segments with linear slopes. The Decay and Release segments are actually exponential decay functions, but are shown linear in these diagrams as a simplification.

1.7.2.1 EG Delay Time

The delay time determines the amount of time in Time Cents from Note-On until the Attack segment begins. Nominally, this time will be set to zero so that no delay in the envelope occurs. If it is not zero, then the envelope generator should hold its output to zero for the specified delay time, then begin the attack segment. Note that the digital oscillator still begins playing the sample at the time the Note-On is received, so this only has the effect of cutting off the start of the sample.



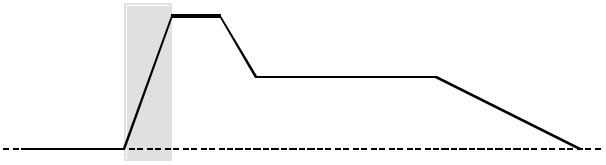
EG Delay Time

1.7.2.2 EG Attack Time

The attack time determines how long the attack segment lasts from the end of the Delay Time until the envelope hits peak level. When the output of the envelope is applied to gain, the Attack segment affects the output of the oscillator in a linear fashion with respect to amplitude. This requires a slightly more complex coupling to the DCA, since the envelope output will undergo a different transform during the attack segment than during all other segments.

When the envelope output is applied to any other destination, the effect is linear with respect to the units used at that particular destination. For example, when applied to pitch, the envelope will always affect the pitch of the oscillator in a linear fashion with respect to units in Pitch Cents, which is exponential with regard to frequency. The filter cutoff F_c will likewise be affected in a linear fashion with respect to Pitch Cents, or exponential with regard to frequency.

If the attack time is set to zero, the envelope generator should immediately go to peak output. Likewise, any controls to which the envelope is routed should also go to full output. For example, if the attack time of the Volume EG is set to zero, the amplitude of the DCA at the time the first sample is output must be at full scale so that any transients in the sample are faithfully reproduced.



EG Attack Time

1.7.2.3 EG Velocity to Attack Scaling

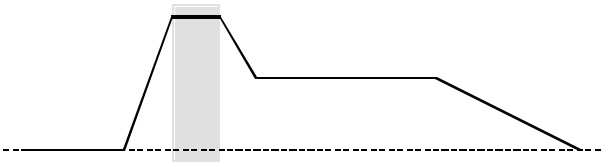
The MIDI velocity scales the duration of the attack segment. Since all times are described in Time Cents, which are logarithmic units, the effect of adding values in Time Cents is multiplicative in actual time. The formula is as follows:

$$Attack_Time_{timecents} = Nominal_Attack_Time_{timecents} + (Velocity / 128) * Attack_Scalar_{timecents}$$

Where *Nominal_Attack_Time* is the attack time constant set by the articulation data, *Velocity* is the MIDI Note-On velocity, and *Attack_Scalar* is the lScale value from the EG Velocity to Attack Time connection.

1.7.2.4 EG Hold Time

After the attack phase, the envelope generator enters the optional Hold phase. If the Hold Time is set to zero, then the generator should proceed directly to the Decay Phase. If the Hold Time is not zero, the generator must hold the generator output level at full scale for the specified time. The Hold Time is specified in Time Cents.

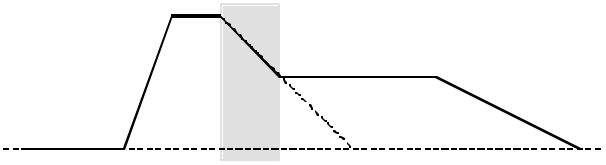


EG Hold Time

1.7.2.5 EG Decay Time

The decay time determines the duration of the decay segment. Because the decay segment is an exponential decay function, it is not possible to express the decay function purely in terms of time. Instead, Decay Time is expressed as the amount of time it takes for the signal to decay from full scale to -96 *dB* in Time Cents.

Since the Sustain Level sets the end point of the Decay segment, the exact duration of the Decay segment will be determined both by the Decay Time, and the Sustain Level. In effect, the Decay Time can be thought of as setting the inverse of the decay rate, as opposed to an absolute time. The dashed line in the following figure indicates how the Decay Time value is actually determined.



EG Decay Time

1.7.2.6 Decay Scaling

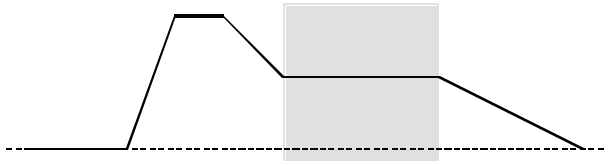
The MIDI note number scales the duration of the decay segment. For example, increasing the note number could decrease the duration of the decay, just as a piano note decays slower for lower notes and faster for higher notes. The formula is as follows:

$$Decay_Time_{timecents} = Nominal_Decay_Time_{timecents} + (Note_Number / 128) * Decay_Scalar_{timecents}$$

Where *Nominal_Decay_Time* is the decay time constant set by the articulation data, *Note_Number* is the MIDI note number, and *Decay_Scalar* is the lScale value from the Mod EG Note to Decay connection.

1.7.2.7 EG Sustain Level

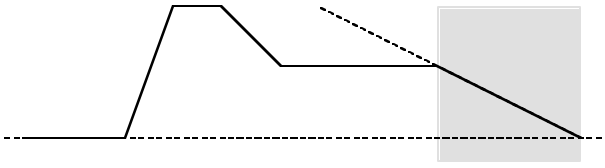
The Decay segment ends when the Sustain Level is reached and the note sustains at this level until the Note-Off event (or Note-On event with a velocity of 0) is received. Raising the Sustain Level will shorten the Decay segment and lengthen the Release segment, while lowering it will have the opposite effect. Sustain is defined as a percentage of the envelope peak in 0.1% increments. Note that Sustain Level is in ordinal units, the actual range is determined by the *lScale* setting of the connection and the unit type is determined by the units used by the destination node. Thus the sustain level as applied to CONN_DST_GAIN is in Gain units (dB), and similarly when applied to CONN_DST_PITCH or CONN_DST_FILTER_CUTOFF is in Pitch Cents.



EG Sustain Level

1.7.2.8 EG Release Time

The release time constant determines the duration of the release segment. Like the Decay time, the Release segment is an exponential decay measured as the time it takes to decay from full scale to –96 *dB*. Just as with the Decay segment, the actual time is dependent on the Sustain Level, or in the case of a Note-Off (or Note-On with a velocity of 0) before the Sustain Level is reached, then the actual time is dependent on the level at the time the Note-Off (or Note-On with a velocity of 0) is received.



EG Release Time

1.7.2.9 EG Shutdown Time

The shutdown time constant determines the duration of the release segment when a note-exclusive or key-exclusive event occurs. It behaves identically to the EG Release Time constant, but is only invoked when an exclusive event causes the shutdown of an oscillator.

1.7.3 Key Number Generator

The Key Number Generator provides a mechanism for modifying the region selection process. It exists for Mobile DLS instruments primarily as a means to accommodate the behavior of the RPN2 Coarse Tuning Parameter. The function is fixed and this connection should not appear in a Mobile DLS file except in a proprietary conditional chunk. Because the Key Number generator affects region selection, input connections to the Key Number Generator are only valid at the global articulation level in the instrument LIST chunk.

The Key Number Generator has a single input DST_KEYNUMBER and a single output SRC_KEYNUMBER. The default input connections route the MIDI note number and RPN2 to DST_KEYNUMBER, allowing for modification of region selection.

The default output connection routes the Key Number Generator to Pitch. There is an implied connection from the output of the Key Number Generator to the control input that selects the correct region. The Key Number to Pitch output connection is allowed at both global and local articulation levels.

1.8 Performance Controllers

MIDI Controllers can be used to modify synthesis parameters in real-time to add depth to a performance. The following MIDI controllers are supported:

1.8.1 Note Number

The MIDI Note Number normally determines the playback frequency of the instrument. The default connection is to map an Equal Temperament 12-tone (ET-12) scale to the keyboard, such that the difference between each key is an ET-12 semitone. However, it is sometimes useful to change the default key scaling so that the pitch does not change across the keyboard. The Mobile DLS Device architecture allows the Note Number to Pitch connection to be specified with a *lScale* of zero, which means that the Note Number will have no effect on the pitch. Other values may be used for *lScale* to modify the tuning for quarter-tones or other useful increments.

1.8.2 Volume

MIDI Volume Controller events (MIDI Controller 7) are connected by default to the DST_GAIN summing node to modify the output volume, and are mapped through the Concave Transform.

1.8.3 Expression

MIDI Expression Controller events (MIDI Controller 11) are connected by default to the DST_GAIN summing node to modify the output volume, and are mapped through the Concave Transform.

1.8.4 Velocity

MIDI Note-on Velocity is connected by default to the DST_GAIN summing node to modify the output volume and is mapped through the Concave Transform. The connection may be scaled as either –96 *dB*, which matches the curve of a typical GM device and is the default setting, or 0 *dB*, which means that Velocity has no effect on the output. The latter is used to simulate organs, harpsichords, and other instruments that do not respond to velocity.

1.8.5 Pan

Only devices that have stereo output are required to support channel panning. Mobile DLS devices are not required to support the panning of individual notes.

After normalization, the MIDI Pan event (MIDI Controller 10) is nominally fed directly to the digitally controlled amplifier where it is used to set the pan position in the stereo field. An equal power equation is used to define the distribution from left to right. DST_PAN is the destination for pan control sources, and is measured in percentage units with a minimum value of –50% and a maximum value of 50%. After all inputs to DST_PAN have been summed, the value should be limited to the range +/-50%.

The equation for individual channel gain in *dB* is given by the following formula:

$$Left\ Channel\ Gain_{db} = 20\ log_{10}\ (cos\ (\pi / 2 * (DST_PAN + 50\%)))$$

$$Right\ Channel\ Gain_{db} = 20\ log_{10}\ (sin\ (\pi / 2 * (DST_PAN + 50\%)))$$

Note that for values of –50% and 50%, one of the two stereo channels will have a gain of negative infinity, meaning that channel should be fully attenuated, while the other will have a gain of 0 *dB*. The following is a graph of the left and right channel outputs plotted against values of the DST_PAN node:

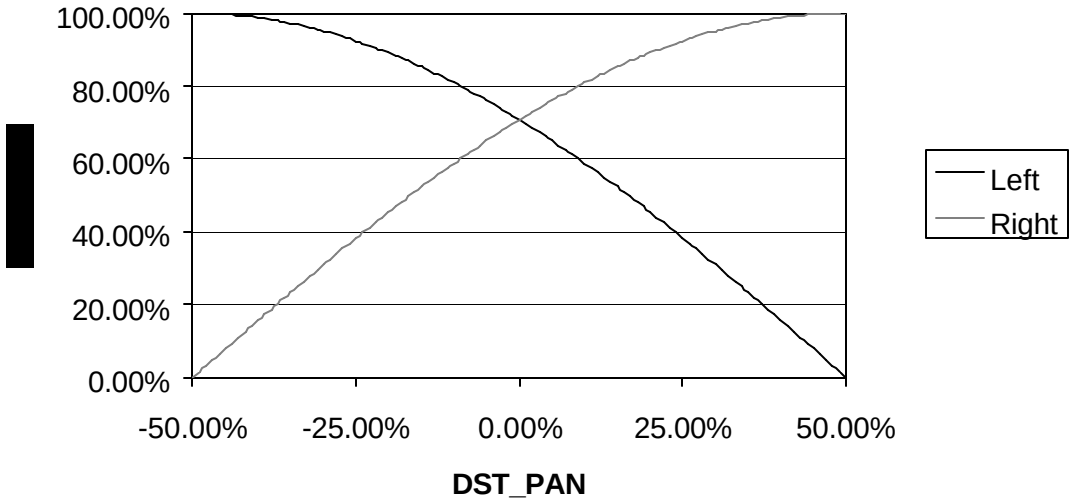


Figure 11 — Pan Curve

A Mobile DLS device that supports stereo must support active panning of a signal within the stereo field without sound artifacts, which will require smoothing filters to the DCA inputs to prevent zipper noise in most implementations. The exact form of these filters is up to the implementer, however there must be no perceptible control lag in the response to MIDI Controller 10 events.

Due to asymmetry in the MIDI controllers, it is impossible to pan a sound to full right unless the *lScale* value applied to DST_PAN is larger than 50%. For this reason, the default connection from CC10 to DST_PAN is 50.8%. When the value of CC10 is 64 the sound will be placed exactly in the center, while values of 0 or 1 will attenuate the right channel completely and a value of 127 will attenuate the left channel completely.

1.8.6 MIDI Controller 1 (Modulation)

The Modulation controller (CC1) may be used to set LFO depth, with separate level assignments for LFO pitch modulation and LFO volume modulation, or filter cutoff frequency modulation.

1.8.7 Channel Pressure

Channel Pressure may be used to set LFO depth, with separate level assignments for LFO pitch modulation, LFO volume modulation, or filter cutoff frequency modulation.

1.8.8 Pitch Bend

The MIDI Pitch Bend event is routed directly to the pitch summing node, and scaled by RPN 0. Registered Parameter Number data entry can be used to set the pitch range.

1.8.9 MIDI Controller 64 (Sustain)

Sustain Pedal MIDI events (MIDI Controller 64) go directly to the control logic, which uses them in conjunction with note on and off events to determine which notes are on.

Sustain Pedal is the same as Hold1 (Damper) in [6,7].

1.8.10 MIDI Controller 91 (Reverb)

Mobile DLS devices do not need to support Reverb, but if supported the following reverb control must be implemented. MIDI Controller 91 events are routed by default to the Reverb effects sends. This determines the amplitude of signal fed to the reverb effects unit. Note that the units for DST_REVERB are in percentage of amplitude, not in *dB* units.

1.8.11 MIDI Controller 93 (Chorus)

Mobile DLS devices do not need to support Chorus, but if supported the following chorus control must be implemented. MIDI Controller 93 events are routed by default to the Chorus effects sends. This determines the amplitude of signal fed to the chorus effects unit. Note that the units for DST_CHORUS are in percentage of amplitude, not in *dB* units.

1.8.12 Registered Parameter Numbers (RPN)

Registered Parameter Numbers (RPNs) for remote data entry are supported, as per the General MIDI specification. The parameters include Fine Tune, Pitch Bend Range, and Coarse Tuning.

1.8.12.1 RPN 0 (Pitch Bend Range)

RPN0 (Pitch Bend Range) is used to alter the range of the pitch bend wheel. A Mobile DLS device must support a maximum pitch bend range of at least +/- 12 semitones.

1.8.12.2 RPN 1 (Fine Tuning)

RPN1 (Fine Tuning) is used to alter the pitch in fine increments of less than a semitone. The control operates in 1/8192 semitone increments and has a range of –8192/8192 to +8191/8192. A Mobile DLS device must have a minimum resolution of 1 cent (1/100th of a semitone) with accuracy of no less than plus or minus 1/2 cent. The default setting is 0, which does not alter the pitch.

Mobile DLS devices do not have to support Fine Tuning on GM drum instruments.

1.8.12.3 RPN 2 (Coarse Tuning)

RPN2 (Coarse Tuning) is used to transpose by semitones. Coarse Tuning alters the MIDI note number before the articulation data, region, and sample selections are made by the Control Logic. This placement prevents unwanted timbre shifts in instruments that have been multisampled. The default setting is 0, which does not alter the pitch.

To accomplish this, the MIDI note number and RPN2 controller inputs are summed together in a block called the Key Number Generator. The output of the Key Number Generator is used to drive the region selection logic as well as set the base frequency of the oscillator prior to any modulations.

Mobile DLS devices do not have to support Coarse Tuning on GM drum instruments.

1.8.13 Registered Parameter Number Data Entry

A Mobile DLS device is required to support the Data Entry MSB and Data Entry LSB Controllers (MIDI Controllers 6 and 38, respectively), but is not required to support the Data Increment and Data Decrement Controllers (MIDI Controllers 96 and 97, respectively).

The following protocol should be used to set RPN values, where *n* is the MIDI channel number, *ww* is the RPN MSB, *xx* is the RPN LSB, *yy* is the Data MSB, and *zz* is the Data MSB (all values are in hexidecimal).

```
Bn 65 ww    // Send RPN MSB
Bn 64 xx    // Send RPN LSB
Bn 06 yy    // Send RPN Data MSB
Bn 26 zz    // Send RPN Data LSB
Bn 65 7F    // Clear RPN MSB
Bn 64 7F    // Clear RPN LSB
```

The last two messages set the RPN to an invalid parameter number, thus preventing an inadvertent change to an RPN value by a subsequent RPN Data MSB or LSB message. If a number of RPN values need to be set, the messages to clear the RPN MSB and LSB may be left to the end of the sequence of messages to reduce the overhead. Note that only the first "B*n*" status byte is necessary due to running status, but all are shown here for clarity.

1.9 Channel Mode Messages

A Mobile DLS device must also support the following Channel Mode messages:

1.9.1 Reset All Controllers (MIDI Controller 121)

The Reset All Controllers message makes use of the data byte to allow for multiple levels of functionality. There are currently two data byte values defined. When the data byte is 0, the device will reset all controllers to their default values except the following controllers:

- Volume (Controller 7)
- Pan Control (Controller 10)

When the data byte is 127, the device will reset all controllers to their power-on default values (see Section 1.11 below). This should include all MIDI controllers and RPNs. It is left to the discretion of the device designer whether to reset any NRPNs used by the device. However, the behavior must be documented.

1.9.2 All Notes Off (MIDI Controller 123)

This message performs a Note-Off event for all notes on the specified MIDI channel. If the Sustain Pedal (MIDI Controller 64) is active, the notes should continue to sustain until a Sustain Pedal release event is sent.

1.9.3 All Sound Off (MIDI Controller 120)

A Mobile DLS device must respond to an All Sound Off message by silencing all notes, that are currently sounding on the specified MIDI channel. Upon receiving the message, all notes should be turned off, and the output set to zero as quickly as possible. Envelopes may be ramped to prevent clicks or pops in the output. This is a panic message, and intended to be used under user control to silence the device in adverse situations. It is not a replacement for sending individual Note-Off events.

1.9.4 Other Channel Mode Messages

It is not required that a Mobile DLS device support the Omni Off, Omni On, Mono On, and Poly On messages, which are MIDI Controller numbers 124 through 127 respectively. If a device does not respond to any of these messages, it should treat them as All Notes Off messages. In all cases, the device response must be documented.

1.10 Active Sensing (0xFE)

A Mobile DLS device is not required to respond to Active Sensing messages.

1.11 Default Values

A Mobile DLS device must reset the following control values during power-on, upon receiving the DLS System Exclusive Turn DLS On message, and when a Reset All Controllers message is received with data byte 127.

Power-On and Reset All Controller Values

- Volume (default = 100)
- Pan Control (default = 64)
- Pitch Bend (default = 0, no pitch change)
- CC100 RPN LSB (default = 127)
- CC91 Reverb Send (default = 40) (if supported)
- Channel Pressure (default = 0)
- RPN1 Fine Tuning (default = 0)
- Default channel priority order (10, 1-9, 11-16)
- Expression (default = 127)
- Modulation (default = 0)
- Sustain Pedal (default = 0)
- CC101 RPN MSB (default = 127)
- CC93 Chorus Send (default = 0) (if supported)
- RPN0 Pitch Bend Range (default = 2)
- RPN2 Coarse Tuning (default = 0)
- Bank address: Channel 10: Bank Select MSB=0x78, LSB=0x00, Program Change=0x00; Other Channels: MSB=0x79, LSB=0x00, Program Change=0x00

1.12 Articulation Architecture

This section defines the set of modules that must be supported and it itemizes a list of required connections between the nodes of these modules. These connections define the flow of signals between modules and ultimately to the digital oscillator, DCF, and DCA to control the pitch and volume of a note as it plays.

1.13 Modulation Routing

The Mobile DLS synthesizer supports the connection blocks listed in Table 5. Each connection lists the *usSource*, *usControl*, *usDestination*, and associated input transforms. The output transform for each of these connections is always TRN_NONE.

The articulation configuration is designed to expand into future specification levels in two ways. First, each module defines a base level set of controls and connectors. Most modules will expand as more controls and connections are added in the future. Secondly, the number of connection blocks will also grow as more articulation modules, control inputs, and required connections are added.

1.13.1 Default Connections

Certain default connections are assumed in the absence of specific data in the instrument. These default connections help reduce the amount of data required to define an instrument, because the default values are generally usable. They also facilitate the development of new instruments, because in the absence of any other articulation data, the default values will produce sound at the expected musical pitch. The sound developer may then further refine the instrument from the default values to suit his or her taste. Table 6 shows the Minimum, Maximum and Unit values for each connection block of the Mobile DLS synthesis architecture.

Table 5 — Modulation Routing

This table contains all the legal connections for a Mobile DLS Synthesizer with fixed routing architecture. The "B" column indicates whether the particular ource connection is Bipolar, and the "I" column indicates whether it is Inverted (these are the Bipolar and Invert bits in the Connection Block).

(*) To improve readability of this table, the "CONN_" prefix has been omitted from these columns.

Articulator Name	UsSource (*)	B	I	Transform	UsControl (*)	B	I	Transform	UsDestination (*)
Modulator LFO									
Mod LFO Frequency	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_LFO_FREQ
Mod LFO Start Delay	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_LFO_DELAY
Vibrato LFO									
Vib LFO Frequency	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_VIB_FREQ
Vib LFO Start Delay	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_VIB_DELAY
Volume EG									
Vol EG Delay Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_DELAYTIME
Vol EG Attack Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_ATTACKTIME
Vol EG Hold Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_HOLDTIME
Vol EG Decay Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_DECAYTIME
Vol EG Sustain Level	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_SUSTAINLEVEL
Vol EG Release Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_RELEASETIME
Vol EG Shutdown Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_SHUTDOWNTIME
Vol EG Velocity to Attack	SRC_KEYONVELOCITY	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_ATTACKTIME
Vol EG Key to Decay	SRC_KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_DECAYTIME
Vol EG Key to Hold	SRC_KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG1_HOLDTIME
Modulator EG									
Mod EG Delay Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_DELAYTIME
Mod EG Attack Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_ATTACKTIME
Mod EG Hold Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_HOLDTIME
Mod EG Decay Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_DECAYTIME
Mod EG Sustain Level	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_SUSTAINLEVEL
Mod EG Release Time	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_RELEASETIME
Mod EG Velocity to Attack	SRC_KEYONVELOCITY	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_ATTACKTIME
Mod EG Key to Decay	SRC_KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_DECAYTIME
Mod EG Key to Hold	SRC_KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_EG2_HOLDTIME
Key Number Generator									
MIDI Note to Key	SRC_MIDINOTE	F	F	Linear	SRC_NONE	F	F	Linear	DST_KEYNUMBER
RPN2 to Key	SRC_RPN2	T	F	Linear	SRC_NONE	F	F	Linear	DST_KEYNUMBER

Articulator Name	UsSource (*)	B	I	Transform	UsControl (*)	B	I	Transform	UsDestination (*)
Filter									
Initial F_c	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_CUTOFF
Initial Q	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_Q
Mod LFO to F_c	SRC_LFO	T	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_CUTOFF
Mod LFO CC1 to F_c	SRC_LFO	T	F	Linear	SRC_CC1	F	F	Linear	DST_FILTER_CUTOFF
Mod LFO Channel Pressure to F_c	SRC_LFO	T	F	Linear	SRC_CHANNEL PRESSURE	F	F	Linear	DST_FILTER_CUTOFF
Mod EG to F_c	SRC_ EG2	F	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_CUTOFF
Velocity to F_c	SRC_KEYONVELOCITY	F	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_CUTOFF
Key Number to F_c	SRC_ KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_FILTER_CUTOFF
Gain									
Vol EG to Gain	SRC_EG1	F	F	Linear	SRC_NONE	F	F	Linear	DST_GAIN
Mod LFO to Gain (v2.1)	SRC_LFO	T	F	Linear	SRC_NONE	F	F	Linear	DST_GAIN
Mod LFO CC1 to Gain. (v2.1)	SRC_LFO	T	F	Linear	SRC_CC1	F	F	Linear	DST_GAIN
Mod LFO Channel Pressure to Gain	SRC_LFO	T	F	Linear	SRC_CHANNEL PRESSURE	F	F	Linear	DST_GAIN
Velocity to Gain	SRC_KEYONVELOCITY	F	T	Concave	SRC_NONE	F	F	Linear	DST_GAIN
MIDI CC7 to Gain	SRC_CC7	F	T	Concave	SRC_NONE	F	F	Linear	DST_GAIN
MIDI CC11 to Gain	SRC_CC11	F	T	Concave	SRC_NONE	F	F	Linear	DST_GAIN
Pitch									
Tuning	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
Pitch Wheel RPN0 to Pitch	SRC_PITCHWHEEL	T	F	Linear	SRC_RPN0	F	F	Linear	DST_PITCH
Key Number to Pitch	SRC_KEYNUMBER	F	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
RPN1 to Pitch	SRC_RPN1	T	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
Vib LFO to Pitch	SRC_VIBRATO	T	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
Vib LFO CC1 to Pitch	SRC_ VIBRATO	T	F	Linear	SRC_CC1	F	F	Linear	DST_PITCH
Vib LFO Channel Pressure to Pitch	SRC_VIBRATO	T	F	Linear	SRC_CHANNEL PRESSURE	F	F	Linear	DST_ PITCH
Mod LFO to Pitch	SRC_LFO	T	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
Mod LFO CC1 to Pitch	SRC_LFO	T	F	Linear	SRC_CC1	F	F	Linear	DST_PITCH
Mod LFO Channel Pressure to Pitch	SRC_ LFO	T	F	Linear	SRC_CHANNEL PRESSURE	F	F	Linear	DST_ PITCH
Mod EG to Pitch	SRC_EG2	F	F	Linear	SRC_NONE	F	F	Linear	DST_PITCH
Output									
Default Pan	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_PAN
MIDI CC10 to Pan	SRC_CC10	T	F	Linear	SRC_NONE	F	F	Linear	DST_PAN
CC91 to Reverb Send	SRC_CC91	F	F	Linear	SRC_NONE	F	F	Linear	DST_REVERB
Default Reverb Send	SRC_NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_REVERB
CC93 to Chorus Send	SRC_CC93	F	F	Linear	SRC_NONE	F	F	Linear	DST_CHORUS
Default Chorus Send	SRC_ NONE	F	F	Linear	SRC_NONE	F	F	Linear	DST_CHORUS

Table 6 — Default Connection Blocks

Mobile DLS Synthesizer Default, Minimum, Maximum and Unit Values for Connection Blocks

Unspecified connection blocks in a connection block list shall be set to the default value shown here by the Mobile DLS synthesis engine.

Articulator	Default Value	Min Value	Max Value	Units
Modulator LFO				
Mod LFO Frequency	5 Hz	0.1 Hz	20 Hz	Absolute Pitch
Mod LFO Start Delay	10 msecs	10 msecs	10 secs	Absolute Time
Vibrato LFO				
Vibrato LFO Frequency	5 Hz	0.1 Hz	20 Hz	Absolute Pitch
Vibrato LFO Start Delay	10 msecs	10 msecs	10 secs	Absolute Time
Vol EG				
Vol EG Delay Time	0 secs	0 secs	40 secs	Absolute Time
Vol EG Attack Time	0 secs	0 secs	40 secs	Absolute Time
Vol EG Hold Time	0 secs	0 secs	40 secs	Absolute Time
Vol EG Decay Time	0 secs	0 secs	40 secs	Absolute Time
Vol EG Sustain Level	100 %	0%	100%	Percent
Vol EG Release Time	0 secs	0 secs	40 secs	Absolute Time
Vol EG Shutdown Time	15 msecs	0 secs	40 secs	Absolute Time
Vol EG Velocity to Attack	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Vol EG Key to Decay	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Vol EG Key to Hold	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Modulator EG				
Mod EG Delay Time	0 secs	0 secs	40 secs	Absolute Time
Mod EG Attack Time	0 secs	0 secs	40 secs	Absolute Time
Mod EG Hold Time	0 secs	0 secs	40 secs	Absolute Time
Mod EG Decay Time	0 secs	0 secs	40 secs	Absolute Time
Mod EG Sustain Level	100%	0%	100%	Percent
Mod EG Release Time	0 secs	0 secs	40 secs	Absolute Time
Mod EG Velocity to Attack	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Mod EG Key to Decay	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Mod EG Key to Hold	0 Time Cents	0 Time Cents	6,000 Time Cents	Relative Time Cents
Key Number Generator				
MIDI Note to Key	12,800 cents	12,800 cents	12,800 cents	Relative Pitch
RPN2 to Key	6,400 cents	0 cents	6,400 cents	Relative Pitch

Articulator	Default Value	Min Value	Max Value	Units
Filter				
Initial F_c	0x7FFFFFFFh	5535 cents	11921 cents	Absolute Pitch
Initial Q	0 *dB*	0 *dB*	22.5 *dB*	Relative Gain
Mod LFO to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Mod LFO CC1 to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Mod LFO Channel Press. to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Mod EG to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Velocity to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Key Number to F_c	0 cents	-12800 cents	12800 cents	Relative Pitch
Gain				
Vol EG to Gain	96 dB	96 dB	96 dB	Relative Gain
Mod LFO to Gain	0 *dB*	0 *dB*	12 *dB*	Relative Gain
Mod LFO CC1 to Gain.	0 *dB*	0 *dB*	12 *dB*	Relative Gain
Mod LFO Chan. Press. to Gain.	0 *dB*	0 *dB*	12 *dB*	Relative Gain
Velocity to Gain	-96 *dB*	0 *dB*	-96 *dB*	Relative Gain
MIDI CC7 to Gain	-96 *dB*	-96 *dB*	-96 *dB*	Relative Gain
MIDI CC11 to Gain	-96 *dB*	-96 *dB*	-96 *dB*	Relative Gain
Pitch				
Tuning	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Pitch Wheel RPN0 to Pitch	12,800 cents	12,800 cents	12,800 cents	Relative Pitch
Key Number to Pitch	12,800 cents	0 cents	12,800 cents	Relative Pitch
RPN1 to Pitch	100 cents	0 cents	100 cents	Relative Pitch
Vib LFO to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Vib LFO CC1 to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Vib LFO Chan. Press. to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Mod LFO to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Mod LFO CC1 to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Mod LFO Chan. Press. to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Mod EG to Pitch	0 cents	-1,200 cents	1,200 cents	Relative Pitch
Output				
Default Pan	0%	-50%	50%	0.1% units
MIDI CC10 to Pan	50.8%	-50.8%	50.8%	0.1% units
Default CC91 to Reverb Send	100%	0%	100%	0.1% units
Default Reverb Send	0%	0%	100%	0.1% units
Default CC93 to Chorus Send	100%	0%	100%	0.1% units
Default Chorus Send	0%	0%	100%	0.1% units

1.14 Data Format Definitions

1.14.1 Absolute Pitch

A signed, 32-bit unit for expressing Absolute Pitch in an exponential form to facilitate modulation in a musical fashion. The formula for converting frequency in Hertz to the DLS Absolute Pitch Unit is as follows:

$$Absolute\ Pitch = (1200 * \log_2 (f / 440) + 6900) * 65536$$

Where f is the frequency of interest. Each absolute pitch unit represents 1/65536 cents.

1.14.2 Relative Pitch

A signed, 32-bit unit for expressing Absolute Pitch in an exponential form to facilitate modulation in a musical fashion. The formula for converting frequency in Hertz to the DLS Absolute Pitch Unit is as follows:

$$Relative\ Pitch = 1200 * \log_2 (f / F) * 65536$$

Where F is the reference frequency and f is the frequency of interest. Each relative pitch unit is 1/65536 cents.

1.14.3 Absolute Time

A signed, 32-bit unit for expressing absolute time in an exponential form to facilitate modulation in a musical fashion. The formula for converting time in seconds to the DLS Absolute Time Unit is as follows:

$$Absolute\ Time = 1200 * \log_2 (time_{secs}) * 65536$$

Since it is impossible to represent zero in this format, the value 80000000h has been chosen to denote an absolute zero. When the value 80000000h is applied to a destination, that destination cannot be modified by other sources; it will always be zero.

1.14.4 Gain

A signed, 32-bit unit for expressing relative signal strengths in an exponential form to facilitate modulation in a musical fashion. Gain is calculated by the following formula:

$$Gain\ Units = 200 * \log_{10} (v / V) * 65536$$

Where V is the reference signal and v is the signal of interest. Positive values indicate gain, while negative values indicate attenuation. Each unit of gain represents 1/655360 *dB*. All references to gain or units of gain in this document refer to amplitude and not power.

1.14.5 Sample Frequency

A 32-bit unsigned integer, with the frequency specified in 1/1000th of Hertz.

1.14.6 Instrument

The instrument header determines the number of regions in an instrument, as well as its bank and program numbers. Each region contains at minimum a <rgnh-ck> region header chunk and a <wlnk-ck> wave link chunk. It may also optionally contain a <wsmp-ck> wave sample chunk, and a <lar2-ck> local articulation chunk.

cRegions Specifies the count of regions for this instrument.

Locale Specifies the MIDI locale (Bank and Program Change) for this instrument.

1.14.7 Region

The Region is used by the Control Logic to decide which sample to use. Each region provides velocity and note ranges to match against an incoming note. If these match, the sample is chosen for performance with the articulation and loop data selected by the region. The Region chunk also contains the Wave Link chunk, and may contain local articulation.

RangeKey Specifies the note range for this region.

RangeVelocity Specifies the velocity range for this region.

fusOptions Specifies flag options for the synthesis of this region. The only flag defined at this time is the *Self Non Exclusive* flag. See Note Exclusivity section for more detail.

usKeyGroup Specifies the key group for a drum instrument. Key group values allow multiple regions within a drum instrument to belong to the same "key group." If a synthesis engine is instructed to play a note with a key group setting and any other notes are currently playing with this same key group, the synthesis engine should turn off all notes with the same key group value as soon as possible. Valid values are:

0	No Key group
1-15	Key groups 1 to 15.
All Others	Reserved

usLayer Optional field for grouping regions into layers. This is intended to facilitate the user interface for editing programs and has no bearing on the synthesis of the sound. Programs may detect the existence of the usLayer field by examining the size of the <rghn-ck>. Programs that ignore *usLayer* must still preserve its value when writing data to a file.

1.14.8 Wave Link

The Wave Link contains the link to the sample data. This is isolated from the region to allow for synthesis methods other than wavetable synthesis. The sample data is indexed through the Pool Table and the other fields provide the information to phase-lock multiple samples together to create stereo or even surround-sound images.

ulChannel Specifies the channel placement of the sample. This is used to place mono sounds within a stereo pair or for multi-track placement. Each bit position within the ulChannel field specifies a channel placement with bit 0 specifying a mono sample or the left channel of a stereo sample. Bit 1 specifies the right channel of a stereo sample.

ulTableIndex Specifies the 0 based index of the cue entry in the wave pool table.

1.14.9 Articulation

The Articulation data specifies a number of flexible connections for routing signals in the synthesizer. The Mobile DLS Device Architecture places a restriction on which connections are valid for different DLS instrument levels. The chunk specifies the number of connections, followed by a list of connections.

cConnectionBlocks Specifies the number (count) of ConnectionBlocks that are contained in the articulator chunk.

usSource Specifies the source for the connection.

usControl Specifies the control for the connection.

usDestination Specifies the destination for the connection

usTransform Specifies the transform used for the connection.

lScale Specifies the scaling value used for the connection.

1.14.10 Wave Sample

The Wave Sample chunk contains the low level parameters for playing a sample. Normally it is contained within the Wave chunk itself, but may also be used at the Region level to override loop points or other parameters.

usUnityNote	Specifies the MIDI note which will replay the sample at original pitch. This value ranges from 0 to 127 (a value of 60 represents Middle C, as defined by the MIDI specification).
sFineTune	Specifies the tuning offset from the *usUnityNote* in 16 bit relative pitch.
lGain	Specifies the gain to be applied to this sample in 32 bit relative gain units. This is used primarily to balance multi-sample splits.
fulOptions	Specifies flag options for the digital audio sample. Flags are defined for disabling 16 bit to 8-bit truncation of samples and compression of samples by the driver.
cSampleLoops	Specifies the number (count) of loop records that are contained in the wave sample chunk. One-shot sounds will have the *cSampleLoops* field set to 0.
ulLoopType	Specifies the loop type.
ulLoopStart	Specifies the start point of the loop in samples as an absolute offset from the beginning of the data in the 'data' chunk of the sample.
ulLoopLength	Specifies the length of the loop in samples.

1.15 Tolerances

Discussions so far have been restricted primarily to the idealized synthesizer. While reducing this synthesizer to a practical working model, certain compromises will undoubtedly be made as a result of design tradeoffs. The purpose of this section is to lay out guidelines for allowable variances from the ideal synthesizer to accommodate tradeoffs made in the design process.

1.15.1 Digital Oscillator

The Digital Oscillator must have a minimum frequency resolution of 1 cent, +/-0.25 cents within the allowable pitch shift range of +2/-4 octaves.

1.15.2 Digitally Controlled Filter

In realizing a practical filter from the ideal filter model, it is necessary to make certain compromises in the implementation. In order to accommodate variances in implementation, the actual filter response is permitted to vary from the ideal filter response described in this document as follows:

- A minimum of 16 discrete resonance values must be supported spanning the range from 0 *dB* to 22.5 *dB*. The discrete resonance values will be approximately linearly spaced (in *dB*) across that range. This corresponds to a step size of 1.5 *dB*. Up to 1.5 *dB* of error is tolerated for any resonance specified within the range 0 *dB* to 22.5 *dB*.
- When the cutoff frequency of a filter is swept with constant resonance, it is typical in many implementations for the actual height of the resonant peak to vary. A deviation of less than 3 *dB* per octave of sweep is required. Note that this is not achieved by simply varying the pole angle. The radius must be varied with the pole angle to achieve a deviation of less than 3 *dB*. The specified resonance in these cases is valid at 1 kHz and must be within 1.5 *dB* of the specified value.
- The maximum required resonance is 22.5 *dB*. Resonance may be specified greater than 22.5 *dB* but support is not guaranteed by all platforms. When the specified resonance is greater than that supported by a device, it will be rendered as the maximum resonance supported.
- The gain at DC must be within +/-0.75 *dB* of the ideal as specified by this document. The gain at all other frequencies must be within +/-1.5 *dB* of the ideal as specified by this document, with F_c of the ideal normalized to the actual F_c implemented by the filter.

- In most implementations, the actual cutoff frequency according to this specification will not exactly correspond to the specified frequency. A deviation of no more than one semitone is required to allow for tracking filters that are attempting to accentuate a particular harmonic.
- When sweeping the filter cutoff, a minimum of 120 discrete cutoff frequencies per octave must be supported. The worst case spacing between any two adjacent discrete values must be no more than 0.1 semitones. Filter stability may be an issue when sweeping the filter cutoff, especially when the step sizes are large. Filters should remain stable with time varying coefficients at a rate of 0.1 semitones per 2 milliseconds when swept from 200Hz to 1200Hz at 0dB, 12dB, and 22.5dB resonance. Theoretical proof may not be possible, so stability should be subjectively evaluated using a 100Hz square wave at -3dB full scale.
- The cutoff frequency according to this specification is not well defined as the resonance frequency approaches one octave below the Nyquist frequency, corresponding to a pole angle of PI/2 radians. In fact, at a pole angle of PI/3 radians and a radius of 0.9, the resonance frequency calculated by this method is almost exactly equal to Nyquist/2, even though the resonant peak is at Nyquist/3. Also, using these parameters, the calculated cutoff frequency intersects the filter response curve at 0 *dB*. As the pole angle is increased, filters specified in this manner change from low-pass to a simple resonator with no attenuation, and finally to a high-pass characteristic (actually only providing high frequency gain). In view of these facts, and that second order filters of this type will not have a dramatic effect on the sound in this case, cutoff frequencies when calculated according to this specification are only required up to $1/6^{th}$ of the sample rate F_s. Higher cutoff frequencies may be specified. However, support is not guaranteed by all platforms. Mobile DLS device requirements support lower output sample frequencies than DLS-2. Due to this difference, Mobile DLS synthesizers operating at lower output sampling rates may have to play instruments where the filter cutoff frequency is within the range from Fs/6 to Fs/2. Therefore, it is required that the filter implementation may not introduce disturbing processing artifacts when the filter cutoff is sweeping beyond the maximum cutoff frequency (Fs/6). When the specified cutoff frequency is greater than that supported by a Mobile DLS device, it will be limited to the maximum supported cutoff frequency. The one exception to this rule is that when the specified cutoff frequency is greater than the Nyquist frequency, the filter will have a flat passband characteristic, regardless of the specified resonance. In other words, the filter will be off in this case. Content developers are advised to specify the maximum value (0x7FFFFFFF) for the filter cutoff when no low pass filter is desired. However, devices will respond as defined in this text.
- Cutoff frequencies must be supported within the range $F_s / 240$ to $F_s / 6$, where F_s is the output sample rate of the device.

1.15.3 Digitally Controlled Amplifier

The Digitally Controlled Amplifier must have a minimum amplitude resolution of 1 *dB*, +/-0.25 *dB* from unity gain down to –85 *dB* in response to external controls and generators. These controls should feed an internal ramp generator that interpolates between control events. The internal resolution of the ramp generator must be no less than 0.0025 *dB* to prevent zipper noise from generators or external controls.

1.15.4 LFO Generator

The LFO generators must be accurate to within 10% of the specified frequency (averaged over 100 cycles) with jitter of no more than 10 milliseconds. Delay times must be accurate to within +/-10 milliseconds.

1.15.5 Envelope Generator

The Envelope Generators must be accurate to within 0.5 *dB*. Timing must be accurate to within +/-10 milliseconds.

1.16 DLS System Exclusive Messages

The following messages are used to control the behavior of the Mobile DLS device. (They are the same messages used in DLS 1 and DLS 2.)

Turn DLS On:

```
F0 7E <Device ID> 0A 01 F7
     F0 7E                  Universal Non-Real Time SysEx header
     <Device ID>            ID of target device (suggest using 7F: Broadcast)
     0A                     sub-ID #1 = DLS message
     01                     sub-ID #2 = DLS On
     F7                     EOX
```

Turn DLS Off:

```
F0 7E < device ID > 0A 02 F7
     F0 7E                  Universal Non-Real Time SysEx header
     <Device ID>            ID of target device (suggest using 7F: Broadcast)
     0A                     sub-ID #1 = DLS message
     02                     sub-ID #2 = DLS Off
     F7                     EOX
```

2. DLS File RIFF Structure

2.1 RIFF Format

DLS instruments are stored in a RIFF form of type 'DLS '. The subchunks of this form are the ‘cdl ‘, ‘vers’, ‘dlid’ 'colh', 'ptbl', ‘ and 'LIST' chunks. There are three top-level LIST chunks: the <lins-list> instrument list chunk contains <ins-list> instrument subchunks; and the <wvpl-list> wave pool chunk contains <wave-list> subchunks (similar to wave files), and the optional <INFO-list> info list chunk contains information about the collection.

The <colh-ck> chunk defines the number of instruments in the collection. The <ptbl-ck> chunk contains a list of reference entries to digital audio data. The <ins-list> subchunks within the <lins-list> chunk are the actual instruments stored in this collection.

The optional chunk <cdl-ck> provides a means for restricting use to compatible devices when used at the top level. The optional <vers-ck> provides a means for identifying the version of the file content. The <dlid-ck> provides a means of managing resources, particularly in an Internet setting.

2.2 RIFF Structure

The following is a RIFF grammar[1] that describes the downloadable instrument collection:

```
<DLS-form>      →    RIFF( ‘DLS ’                          // Collection
                     [<cdl-ck>]
                     [<vers-ck>]
                     [<dlid-ck>]
                     <colh-ck>
                     <lins-list>
                     <ptbl-ck>
                     <wvpl-list>
                     [<INFO-list>]
                )

<wvpl-list>     →    LIST( ‘wvpl’ <wave-list>...)          // Wave Pool

<wave-list>     →    LIST( ‘wave’                          // Wave File
                     [<dlid-ck>]
                     <fmt-ck>
                     <data-ck>
                     [<wsmp-ck>]
                     [<INFO-list>]
                )
```

[1] See Microsoft Windows Multimedia Programmer’s Reference

```
<lins-list>      →     LIST( ‘lins’ <ins-list>...)              // List of Instruments

<ins-list>       →     LIST( ‘ins ‘                             // Instrument
                       [<dlid-ck>]
                       <insh-ck>
                       <lrgn-list>
                       [<lart-list>]
                       [<lar2-list>]
                       [<INFO-list>]
                 )

<lrgn-list>      →     LIST( ‘lrgn’ <rgn-list> | <rgn2-list>... ) // Region list

<rgn-list>       →     LIST('rgn '                              // L1 Region
                       [<cdl-ck>]
                       <rgnh-ck>
                       <wsmp-ck>
                       [<wlnk-ck>]
                       [<lart-list>]
                       [<lar2-list>]
                       [<INFO-list>]
                 )

<rgn2-list>      →     LIST('rgn2'                              // L2 Region
                       [<cdl-ck>]
                       <rgnh-ck>
                       <wsmp-ck>
                       [<wlnk-ck>]
                       [<lar2-list>]
                       [<INFO-list>]
                 )

<lart-list>      →     LIST( ‘lart’ [<cdl-ck>], <art1-ck>... )  // List of Articulators

<lar2-list>      →     LIST( ‘lar2’ [<cdl-ck>], <art2-ck>... )  // List of Articulators

<INFO-list>      →     LIST( ‘INFO’ <info_text-ck>...)
```

Table 7 — RIFF Definitions

RIFF Chunk	**Definition**
<colh-ck>	A DLS collection header as defined later in this document.
<dlid-ck>	A globally unique identifier chunk as defined later in this document.
<cdl-ck>	A conditional chunk as defined later in this document.
<insh-ck>	An instrument header chunk as defined later in this document.
<rgnh-ck>	A region header chunk as defined later in this document.
<art1-ck>	A DLS-1 articulator data chunk as defined later in this document.
<art2-ck>	A DLS-2 articulator data chunk as defined later in this document.
<wlnk-ck>	A wave link chunk as defined later in this document.
<wsmp-ck>	A wave sample chunk as defined later in this document.
<ptbl-ck>	A pool table data chunk as defined later in this document.
<vers-ck>	An optional version chunk as defined later in this document.
<wave-list>	A DLS wave file chunk.
<info_text-ck>	A text chunk within an <INFO-list> chunk as defined later in this document.

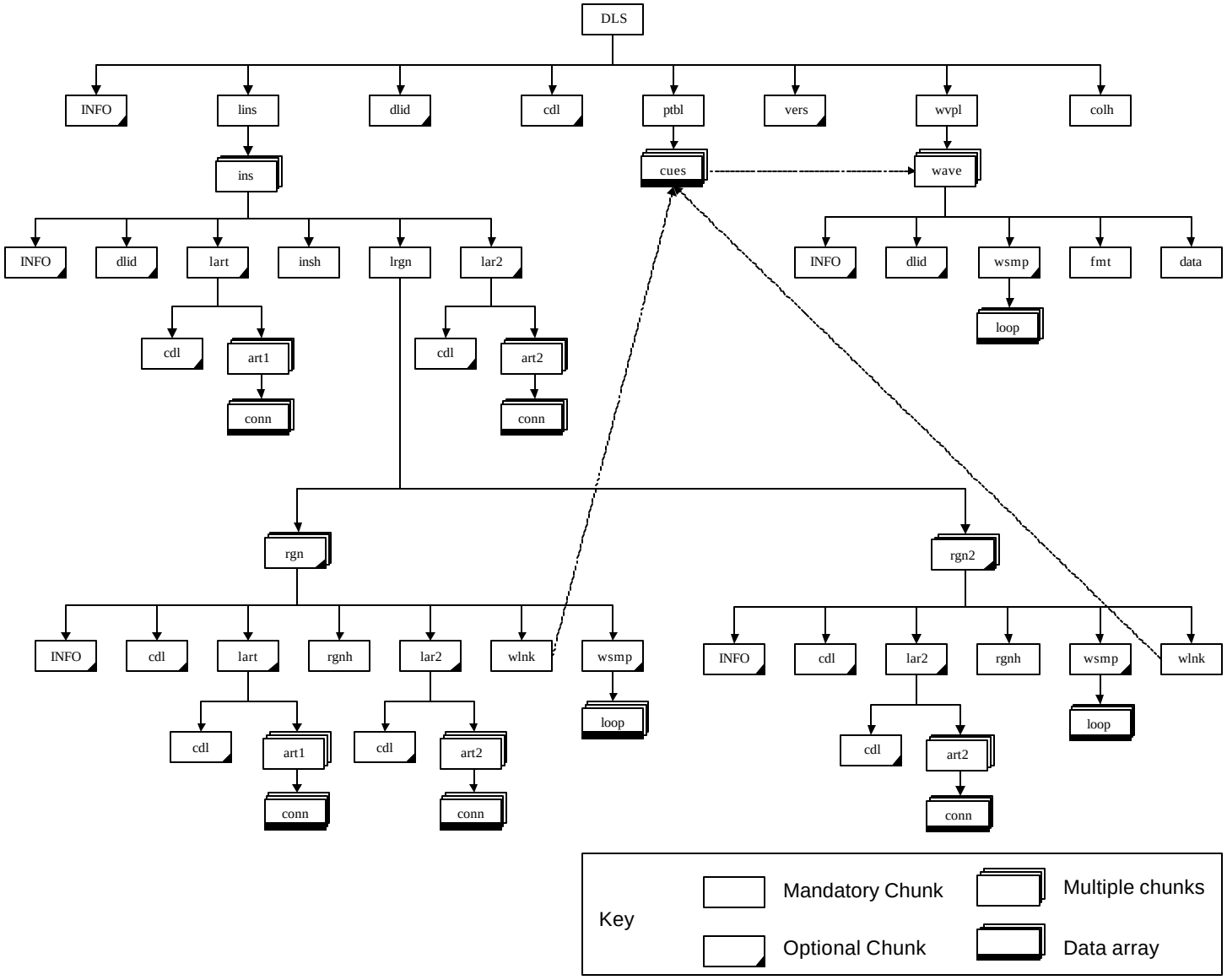


Figure 12 — DLS File Format

2.3 LIST Chunk

A LIST chunk contains a list, or ordered sequence, of subchunks. A LIST chunk is defined as follows:

```
LIST( <list-type> [<chunk>]... )
```

The <list-type> is a four-character code that identifies the contents of the list.

If an application recognizes the list type, it should know how to interpret the sequence of subchunks. However, since a LIST chunk may contain only subchunks (after the list type), an application that does not know about a specific list type can still walk through the sequence of subchunks.

Like chunk IDs, list types must be registered, and an all-lowercase list type has meaning relative to the form that contains it.

2.4 <colh-ck>, Collection Header Chunk

The <colh-ck> collection header defines an instrument collection. The <colh-ck> is defined as follows:

```
<colh-ck>    →          colh(  <cInstruments:ULONG>  )
```

Field	Description
cInstruments	Specifies the count of instruments in this collection, and indicates the number of <ins-ck> instrument chunks in the 'lins' list. This enables the file parser to pre-allocate memory for storage of instruments. If the *cInstruments* field does not match the 'lins' count, the DLS file is in error. It is recommended that sound development tools allow the user to ignore this error and correct the *cInstruments* field to reflect the actual number of instruments in the collection.

A <colh-ck> chunk is typically (but not necessarily) the second chunk within the enclosing <DLS-form> collection chunk.

2.5 <dlid-ck>, DLSID Chunk

The <dlsd-ck> defines an optional globally unique identifier (DLSID) for a complete <DLS-form> or for an element within it. A DLSID is identical to an OSF/DCE 'UUID' (also known as a Microsoft® 'GUID', 'CLSID', 'IID' or 'FMTID'). The <dlid-ck> is defined as follows:

```
<dlid-ck>    →          dlid( <DLSID> )
<DLSID>      →          struct
                        {
                        ULONG       ulData1;
                        USHORT      usData2;
                        USHORT      usData3;
                        BYTE        abData4[8];
                        }
```

The <dlid-ck> chunk:

Field	Description
DLSID	Specifies a 128-bit (16 byte) integer used as a globally unique identifier for a content element. The DCE standard specifies that the string representation for a UUID contains five fixed-length, hyphen-delimited groups of hexadecimal digits, grouped 8-4-4-4-12, as shown: 60DF3430-0266-11cf-BAA6-00AA003E0EED

The <DLSID> structure:

Field	Description
ulData1	Contains the binary value of the first group of the string value (8 hexadecimal digits).
usData2	Contains the binary value of the second group of the string value (4 hexadecimal digits).
usData3	Contains the binary value of the third group of the string value (4 hexadecimal digits).
abData4[8]	Contains the fourth and fifth groups of the string value. Array elements 0 and 1 of hold the fourth group (4 digits), while elements 2 through 7 hold the final group (12 digits).

The C declaration for the example DLSID would be:

```
const DLSID example =
{ 0x60df3430, 0x0266, 0x11cf, { 0xba, 0xa6, 0x00, 0xaa, 0x00, 0x3e, 0x0e, 0xed } };
```

The actual hexadecimal byte sequence (in ascending order) would be:

```
30 34 DF 60    66 22    CF 11    BA A6 00 AA 00 3E 0E ED
```

Byte order in the first three fields appears reversed because Intel® (little-endian) binary representation is used.

The DLSID value must be generated *by computer and never manually*, in order to obtain a unique value (software for this purpose is commonly available). The DLSID must be generated using the UUID algorithm specified by OSF DCE[2], which uses a combination of the following information to generate the value:

- • The current date and time
- • A *clock sequence* and related persistent state to deal with retrograde motion of clocks
- • A forcibly incremented counter to deal with high-frequency allocations
- • A globally unique IEEE machine identifier, obtained from a network card. (If no network card is present, a machine identifier may be synthesized from highly variable machine states and stored persistently).

A DLSID uniquely designates a particular DLS resource (collection, instrument, wave file, or capability). If the resource is modified in any way *whatsoever*, a new DLSID must be generated and attached to that resource. Therefore, changing an instrument or wave file also requires changing the DLSID for the enclosing collection.

[2] Chapter 10, "DEC/HP Network Computing Architecture Remote Procedure Call RunTime Extensions Specification Version OSF TX1.0.11", Steven Miller, July 23 1992.

DLSIDs provide four principal benefits:

- They allow DLS Devices to detect multiply-referenced resources and share a single copy of the data.
- They allow an application to determine whether a DLS resource located on a remote network site is already present on the local system, before downloading the resource.
- They facilitate searching for a particular resource and building special-purpose collections.
- They support error recovery. A DLS manager applet can identify and recover "orphan" resources left in a locked state by a faulty application.
- They provide a way of uniquely identifying various capabilities of the device, which combined with the Conditional Chunk, facilitates forward- and backward-compatibility of instrument collections.

DLSID chunks are not required in DLS files.

2.6 <cdl-ck>, Conditional Chunk

The <cdl-ck> conditional chunk is an optional chunk that can only be found in a list and defines the set of conditions under which the associated chunks in the list are to be used. The <cdl-ck> is defined as follows:

```
<cdl-ck>     →          cdl (  <Operation>
                               [[<Operation>]…]
                           )

<Operation>  →          USHORT      Opcode;
             -or-
             struct
             {
                        USHORT      Opcode;
                                    DLSID       Query;
                        }
             -or-
             struct
             {
                                    USHORT      Opcode;
                                    ULONG       Constant;
                        }
```

The <cdl-ck> consists of a list of opcodes, which may optionally be followed by: a DLSID (GUID) which represents a query to the DLS device; or by a 32-bit unsigned integer constant. The <cdl-ck> creates an extendable, reverse-polish-notation (RPN) grammar for querying a device. On the basis of the query results, the file parser will then either use or discard the contents of the list within which the <cdl-ck> is contained. It is recommended, but not required, that the <cdl-ck> appear as the first subchunk in the list to allow the parser to decide whether to parse or ignore the list contents prior to actually parsing any subchunks in the list.

In the event that a conditional chunk exists at the top level of the DLS form, and the DLS device fails the conditional tests specified within that chunk, the entire DLS file is to be ignored. In this event, an error code should be returned to the application code through the driver interface indicating that the device does not support the required features

set for this file. It is recommended that development tools such as sound editors allow the user to ignore such an error and proceed with the download as a means of porting sounds from one device type to another.

All Mobile DLS devices must support a minimum of eight (8) 32-bit words of accumulator stack for storing temporary values. An opcode may designate an unsigned integer constant, a device query, or a math operation. In the case of a DLS_CDL_CONST opcode, the supplied value is pushed onto the accumulator stack. In the case of the DLS_CDL_QUERY opcode, the device is queried, and the return value is pushed onto the accumulator stack. If the DLSID query is not supported, a FALSE value is pushed onto the stack.

In the case of the DLS_CDL_QUERY_SUPPORTED opcode, the device is queried and if it supports the query, a TRUE value is pushed on the stack, otherwise a FALSE value is pushed on the stack. Note that the DLS_CDL_QUERY_SUPPORTED opcode requires that interface to the device make the distinction between a query that is unsupported, and a query that returns FALSE.

In the case of a unary operator such as DLS_CDL_NOT, the value on the top of the stack is popped off, logically inverted, and then pushed back on the stack. In the case of a binary operator, such as DLS_CDL_GT, the two values on the top of the stack are popped off, compared, and the logical result is pushed onto the stack.

When the file parser reaches the end of the <cdl-ck> conditional chunk, the value on the top of the stack is examined, and if it evaluates as logically true, the list that contains the conditional chunk is parsed. If the value is false, the entire list is discarded.

Logical results are represented as follows: FALSE equates to all zeroes, TRUE equates to all ones. However, any non-zero value used as a logical result will be evaluated as TRUE. Therefore the DLS_CDL_NOT opcode must change any non-zero value to zero.

The follow table describes the opcodes that are valid in a <cdl-ck>. X is assumed to be the value at the top of the stack, Y is assumed to be the previous value on the stack, and Z is a 32-bit constant that only follows the DLS_CDL_CONST opcode. The results of the operation are pushed onto the stack.

Opcode	**Value**	**Equivalent C Code**
DLS_CDL_AND	0x0001	`X = X & Y`
DLS_CDL_OR	0x0002	`X = X | Y`
DLS_CDL_XOR	0x0003	`X = X ^ Y`
DLS_CDL_ADD	0x0004	`X = X + Y`
DLS_CDL_SUBTRACT	0x0005	`X = X - Y`
DLS_CDL_MULTIPLY	0x0006	`X = X * Y`
DLS_CDL_DIVIDE	0x0007	`X = X / Y`
DLS_CDL_LOGICAL_AND	0x0008	`X = X && Y`
DLS_CDL_LOGICAL_OR	0x0009	`X = X || Y`
DLS_CDL_LT	0x000A	`X = (X < Y)`
DLS_CDL_LE	0x000B	`X = (X <= Y)`
DLS_CDL_GT	0x000C	`X = (X > Y)`
DLS_CDL_GE	0x000D	`X = (X >= Y)`
DLS_CDL_EQ	0x000E	`X = (X == Y)`
DLS_CDL_NOT	0x000F	`X = !X`
DLS_CDL_CONST	0x0010	`X = Z`
DLS_CDL_QUERY	0x0011	`X = Query(DLSID) (see text)`
DLS_CDL_QUERY_SUPPORTED	0x0012	`X = QuerySupported(DLSID) (see text)`

The DLS_CDL_QUERY and DLS_CDL_QUERY_SUPPORTED opcodes are followed by a DLSID that uniquely describes the feature to be queried. Additional DLSID's may be defined by device manufacturers using the specified algorithm to describe features proprietary to that device. NOTE: *It is important that all proprietary DLSID's be generated according to the prescribed algorithm to guarantee that unique ID's are generated.*

All Mobile DLS devices must support the following queries:

DLSID Query	**Description**
DLSID_SupportsMobileDLS	Returns TRUE if device supports Mobile DLS
DLSID_SupportsMobileDLSOptionalBlocks[3]	Returns TRUE if device supports Mobile DLS and the voice architecture includes DCF and Vibrato LFO.
DLSID_SupportsDLS1	Returns TRUE if device supports DLS-1
DLSID_SupportsDLS2	Returns TRUE if device supports DLS-2
DLSID_ManufacturersID	Returns MIDI Manufacturers ID in low 24 bits.
DLSID_ProductID	Returns Product ID (manufacturer specific)
DLSID_GMInHardware	Returns TRUE if device supports General MIDI in hardware
DLSID_SampleMemorySize	Returns the amount of memory available in the device in samples
DLSID_SamplePlaybackRate	Returns the true sample playback rate of the device in Hertz
DLSID_GSInHardware	Returns TRUE if device supports GS in hardware
DLSID_XGInHardware	Returns TRUE if device supports XG in hardware

The numeric values for the above DLSID's can be found in the DLS-2 or Mobile DLS Header File in this document.

2.6.1 DLS-1 and DLS-2 Compatibility

Through the use of the Conditional Chunk, it is possible to create an instrument collection that is fully compatible with a DLS-1 Device, while also taking advantage of features of a DLS-2 Device.

Since the DLS-1 Device does not support the <cdl-ck> Conditional Chunk, but is required to ignore any chunk it does not support, it is safe to include the <cdl-ck> chunk in an <lart-ck>chunk, <rgn-ck> chunk, or <lrgn-ck> list. The following conditional chunk will act as a sentinel to prevent a DLS-2 device from accessing chunks intended for a DLS-1 device, while simultaneously allowing a DLS-1 device to freely access them:

```
cdl(        DLS_CDL_QUERY(DLSID_SupportsDLS2)
            DLS_CDL_NOT
)
```

The DLS-2 device will recognize the query and respond TRUE, causing the file parser to ignore the DLS-1 blocks. Data intended for the DLS-2 device can then be placed inside 'lar2' or 'rgn2' chunks and thus isolated from the DLS-1 device, which will ignore those chunks. Adding a conditional chunk that requires support for DLS-2 further isolates the blocks from possible future incompatibilities:

```
cdl(        DLS_CDL_QUERY(DLSID_SupportsDLS2))
)
```

Finally, adding the following conditional chunk at the beginning of the file will help establish the overall compatibility of the file:

```
cdl(        DLS_CDL_QUERY(DLSID_SupportsDLS1)
      DLS_CDL_QUERY(DLSID_SupportsDLS2)
DLS_CDL_LOGICAL_OR
)
```

[3] If the device detects that the instrument can be played correctly on the available voice architecture, it is free to ignore this conditional.

The above chunk requires that the device support either DLS-1 or DLS-2 in order to continue processing the file. As noted previously, development tools should have a mode that allows the user to override the conditional chunk as a means of moving sounds between otherwise incompatible devices.

There are several methods that can be used to create files that take advantage of DLS-2 features when available while still maintaining compatibility with DLS-1 devices. One method involves creating a completely separate set of <rgn2-ck> chunks with data specific to the DLS-2 device, and putting a 'NOT DLS-2' conditional chunk on the <rgn-ck> chunks to prevent the DLS-2 device from trying to access DLS-1 chunks. This also allows for different samples to be played for DLS-1 and DLS-2 devices. This allows the sound designer the opportunity to take advantage of the resonant filters in the DLS-2 device, and use a sampled filter sweep in the DLS-1 device with slightly less satisfactory results, but still compatible.

Another method is to simply augment the DLS-1 data with DLS-2 articulation data. This can be accomplished by putting the basic synthesis parameters into the <art1-ck> chunk, and putting only the DLS-2 extensions in the <art2-ck> chunk.

2.7 <insh-ck>, Instrument Header Chunk

The <insh-ck> defines an instrument within a collection. The <insh-ck> is defined as follows:

```
<insh-ck>   →   insh(
                <cRegions:ULONG>
                <Locale:MIDILOCALE>
                )
```

The <insh-ck> chunk:

Field	Description
cRegions	Specifies the count of regions for this instrument.
Locale	Specifies the MIDI locale for this instrument.

Additional Structure Definitions:

```
typedef struct _MIDILOCALE
{
ULONG ulBank;
ULONG ulInstrument;
} MIDILOCALE, *MIDILOCALE;
```

Field	Description
ulBank	Specifies the MIDI bank location. Bits 0-6 are defined as MIDI CC32 and bits 8-14 are defined as MIDI CC0. Bits 7 and 15-30 are reserved and should be written to zero. If the **F_INSTRUMENT_DRUMS** flag (Bit 31) is equal to 1 then the instrument is a drum instrument; if equal to 0 then the instrument is a melodic instrument. A Mobile DLS sound engine shall ignore Bit 31. Mobile DLS content should set the Bit 31 according to DLS 2.1.
ulInstrument	Specifies the MIDI Program Change (PC) value. Bits 0-6 are defined as PC value and bits 7-31 are reserved and should be written to zero.

An <insh-ck> instrument header is typically (but not necessarily) the ***second*** chunk within the enclosing <ins-list> instrument chunk. Other chunks at the same nesting level include a <lrgn-list> list of regions chunk, and an optional <lart-list> list of articulators chunk (for melodic instruments only).

2.8 <rgnh-ck>, Region Header Chunk

The <rgnh-ck> defines a region within an instrument. The <rgnh-ck> is defined as follows:

```
<rgnh-ck>        →          rgnh(
                              <RangeKey:RGNRANGE>
                              <RangeVelocity:RGNRANGE>
                              <fusOptions:USHORT>
                              <usKeyGroup:USHORT>
                              [<usLayer:USHORT>]
                            )
```

The <rgnh-ck> chunk:

Field	Description
RangeKey	Specifies the key range for this region.
RangeVelocity	Specifies the velocity range for this region.
fusOptions	Specifies flag options for the synthesis of this region. Current options are: **F_RGN_OPTION_SELFNONEXCLUSIVE** This option specifies that if a second Note-On of the same note is received by the synthesis engine, then the second note will be played as well as the first. This option is off by default and the synthesis engine will force a Note-Off of the prior note if a second note is received of the same value.
usKeyGroup	Specifies the key group for a drum instrument. Key group values allow multiple regions within a drum instrument to belong to the same "key group." If a synthesis engine is instructed to play a note with a key group setting and any other notes are currently playing with this same key group, then the synthesis engine should turn off all notes with the same key group value as soon as possible. Valid values are: 0 = no key group Valid key groups are 1 to 15. All Others Reserved

usLayer[Optional] Indicates the layer of this region for editing purposes. This field facilitates the organization of overlapping regions into layers for display to the user of a DLS sound editor. For example, if a piano sound and a string section are overlapped to create a piano/string pad, all the regions of the piano might be labeled as layer 1, and all the regions of the string section might be labeled as layer 2. The program must read the <rgnh-ck> chunk size to determine whether the *usLayer* field exists in the chunk. Programs that do not support the *usLayer* field should preserve the value and write it back to the file.

0 = no layer information

non-zero = valid layer

Additional Structure Definitions:

```
typedef struct _RGNRANGE
     {
USHORT usLow;     /* Low Value of Range */
USHORT usHigh;    /* High Value of Range*/
} RGNRANGE, *RGNRANGE;
```

A <rgnh-ck> region header is typically (but not necessarily) the ***first*** chunk within the enclosing <rgn-list> region chunk. Other chunks at the same nesting level include a <wsmp-ck> wave sample chunk, a <wlnk-ck> wave link chunk, and an optional <lart-list> list of articulators chunk (for drum instruments only).

2.9 <art1-ck>, DLS-1 Articulator Chunk

The <art1-ck> articulator chunk specifies parameters which modify the playback of a sample used in DLS-1 downloadable instruments. The <art1-ck> chunk may exist at either the instrument level, in which case it contains global articulation data, or at the region level, in which case it contains local articulation data. However, the DLS-1 architecture restricts melodic instruments to only global articulation data and restricts drum instruments to only local articulation data.

The <art1-ck> chunk is defined as follows:

```
<art1-ck>    →          art1(
                        <cbSize:ULONG>
                        <cConnectionBlocks:ULONG>
                        <ConnectionBlock(s)>...
                        )

<ConnectionBlock> →     struct
                        {
                            USHORT   usSource;
                            USHORT   usControl;
                            USHORT   usDestination;
                            USHORT   usTransform;
                            LONG     lScale;
                        }
```

The <art1-ck> chunk:

Field	Description
cbSize	Specifies the size of the structure in bytes. *This size does not include the connection blocks*. This field is needed to distinguish the amount of data in the structure versus the list of connections and allow for additions to this structure in the future. This cannot be determined from the chunk size.

cConnectionBlocks	Specifies the number (count) of <ConnectionBlock> records that are contained in the <art2-ck> articulator chunk. The <ConnectionBlock> records are stored immediately following the **cConnectionBlocks** data field.

The <ConnectionBlock> structure:

Field	Description
usSource	Specifies the source for the connection.
usControl	Specifies the control for the connection.
usDestination	Specifies the destination for the connection
usTransform	Specifies the input and output transforms used for the connection.
lScale	Specifies the scaling value used for the connection.

The list of connection blocks defines both the architecture and settings for an instrument or instrument region. Although this could be defined as a simple structure of values, the connection graph model allows for future chunk types which will have a much greater possible number of connections, making the use of a structure unwieldy.

By using the connection graph, a single model can be used for future architectures.

Table 8 lists the defined sources, controls, destinations, and transforms of a DLS-1 synthesizer:

Table 8 — DLS-1 Sources, Controls, Destinations, and Transforms

Modulator Sources		
0x0000	CONN_SRC_NONE	No Source
0x0001	CONN_SRC_LFO	Low Frequency Oscillator
0x0002	CONN_SRC_KEYONVELOCITY	Note-On Velocity
0x0003	CONN_SRC_KEYNUMBER	Note Number
0x0004	CONN_SRC_EG1	Envelope Generator 1
0x0005	CONN_SRC_EG2	Envelope Generator 2
0x0006	CONN_SRC_PITCHWHEEL	Pitch Wheel
MIDI Controller Sources		
0x0081	CONN_SRC_CC1	Modulation
0x0087	CONN_SRC_CC7	Channel Volume
0x008a	CONN_SRC_CC10	Pan
0x008b	CONN_SRC_CC11	Expression
Registered Parameter Numbers		
0x0100	CONN_SRC_RPN0	RPN0 - Pitch Bend Range
0x0101	CONN_SRC_RPN1	RPN1 - Fine Tune
0x0102	CONN_SRC_RPN2	RPN2 - Coarse Tune
Generic Destinations		
0x0000	CONN_DST_NONE	No Destination
0x0001	CONN_DST_GAIN	Gain
0x0002	CONN_DST_RESERVED	Reserved – DO NOT USE
0x0003	CONN_DST_PITCH	Pitch
0x0004	CONN_DST_PAN	Pan
Modulator LFO Destinations		
0x0104	CONN_DST_LFO_FREQUENCY	LFO Frequency
0x0105	CONN_DST_LFO_STARTDELAY	LFO Start Delay Time
EG Destinations		
0x0206	CONN_DST_EG1_ATTACKTIME	EG1 Attack Time
0x0207	CONN_DST_EG1_DECAYTIME	EG1 Decay Time
0x0208	CONN_DST_EG1_RESERVED	Reserved – DO NOT USE
0x0209	CONN_DST_EG1_RELEASETIME	EG1 Release Time
0x020A	CONN_DST_EG1_SUSTAINLEVEL	EG1 Sustain Level
0x030A	CONN_DST_EG2_ATTACKTIME	EG2 Attack Time
0x030B	CONN_DST_EG2_DECAYTIME	EG2 Decay Time
0x030C	CONN_DST_EG2_RESERVED	Reserved – DO NOT USE
0x030D	CONN_DST_EG2_RELEASETIME	EG2 Release Time
0x030E	CONN_DST_EG2_SUSTAINLEVEL	EG2 Sustain Level
Transforms		
0x0000	CONN_TRN_NONE	No Transform
0x0001	CONN_TRN_CONCAVE	Concave Transform

2.10 <art2-ck>, DLS-2 Articulator Chunk

The <art2-ck> articulator chunk specifies parameters which modify the playback of a sample used in DLS-2 downloadable instruments. The <art2-ck> chunk may exist at either the instrument level, in which case it contains global articulation data, or at the region level, in which case it contains local articulation data. The <art2-ck> chunk is defined as follows:

```
<art2-ck>    →          art2(
                        <cbSize:ULONG>
                        <cConnectionBlocks:ULONG>
```

```
                                    <ConnectionBlock(s)>...
                                    )

<ConnectionBlock> →                 struct
                                    {
                                            USHORT          usSource;
                                            USHORT          usControl;
                                            USHORT          usDestination;
                                            USHORT          usTransform;
                                            LONG            lScale;
                                    }
```

The <art2-ck> chunk:

Field	Description
cbSize	Specifies the size of the structure in bytes. *This size does not include the connection blocks.* This field is needed to distinguish the amount of data in the structure versus the list of connections and to allow for additions to this structure in the future. This cannot be determined from the chunk size.
cConnectionBlocks	Specifies the number (count) of <ConnectionBlock> records that are contained in the <art2-ck> articulator chunk. The <ConnectionBlock> records are stored immediately following the **cConnectionBlocks** data field.

The <ConnectionBlock> structure:

Field	Description
usSource	Specifies the source for the connection.
usControl	Specifies the control for the connection.
usDestination	Specifies the destination for the connection
usTransform	Specifies the input and output transforms used for the connection.
lScale	Specifies the scaling value used for the connection.

The *usTransform* field contains information that indicates the type of transforms that apply to the *usSource*, *usControl* source inputs and the also the output transform. The fields are as follows:

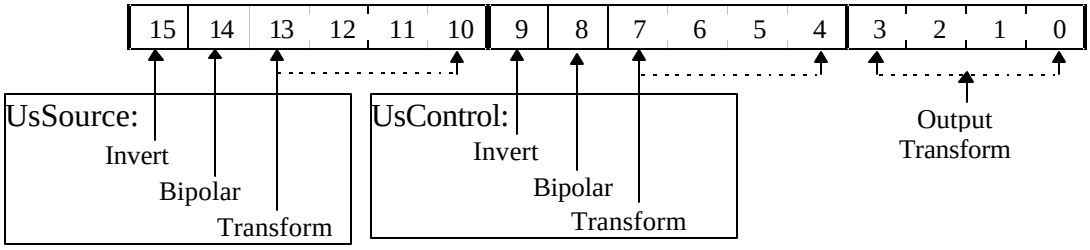


Figure 13 — usTransform Field

Bits 0-3 specify one of 16 possible output transforms. Bits 4-7 specify one of 16 possible transforms to apply to the *usControl* input. Bits 8 and 9 specify whether the *usControl* input should be inverted and/or bipolar. Bits 10-13 specify one of 16 possible transforms to apply to the *usSource* input. Bit 14 and 15 specify whether the *usSource* input should be inverted and/or bipolar.

The list of connection blocks defines both the architecture and settings for an instrument or instrument region. Although this could be defined as a simple structure of values, the connection graph model allows for future chunk types which will have a much greater possible number of connections, making the use of a structure unwieldy.

By using the connection graph, a single model can be used for future architectures.

Table 9 lists the defined sources, controls, destinations, and transforms of a DLS-2 synthesizer.

Table 9 — DLS-2 Sources, Controls, Destinations, and Transforms

Modulator Sources		
0x0000	CONN_SRC_NONE	No Source
0x0001	CONN_SRC_LFO	Low Frequency Oscillator
0x0002	CONN_SRC_KEYONVELOCITY	Note-On Velocity
0x0003	CONN_SRC_KEYNUMBER	Note Number
0x0004	CONN_SRC_EG1	Envelope Generator 1
0x0005	CONN_SRC_EG2	Envelope Generator 2
0x0006	CONN_SRC_PITCHWHEEL	Pitch Wheel
0x0007	CONN_SRC_POLYPRESSURE	Polyphonic Pressure
0x0008	CONN_SRC_CHANNELPRESSURE	Channel Pressure
0x0009	CONN_SRC_VIBRATO	Vibrato LFO
MIDI Controller Sources		
0x0081	CONN_SRC_CC1	Modulation
0x0087	CONN_SRC_CC7	Channel Volume
0x008a	CONN_SRC_CC10	Pan
0x008b	CONN_SRC_CC11	Expression
0x00db	CONN_SRC_CC91	Chorus Send
0x00dd	CONN_SRC_CC93	Reverb Send
Registered Parameter Numbers		
0x0100	CONN_SRC_RPN0	RPN0 - Pitch Bend Range
0x0101	CONN_SRC_RPN1	RPN1 - Fine Tune
0x0102	CONN_SRC_RPN2	RPN2 - Coarse Tune
Generic Destinations		
0x0000	CONN_DST_NONE	No Destination
0x0001	CONN_DST_GAIN	Gain
0x0002	CONN_DST_RESERVED	Reserved
0x0003	CONN_DST_PITCH	Pitch
0x0004	CONN_DST_PAN	Pan
0x0005	CONN_DST_KEYNUMBER	Key Number Generator
Channel Output Destinations		
0x0010	CONN_DST_LEFT	*Left Channel Send
0x0011	CONN_DST_RIGHT	*Right Channel Send
0x0012	CONN_DST_CENTER	*Center Channel Send
0x0013	CONN_DST_LFE_CHANNEL	*LFE Channel Send
0x0014	CONN_DST_LEFTREAR	*Left Rear Channel Send
0x0015	CONN_DST_RIGHTREAR	*Rt Rear Channel Send
0x0080	CONN_DST_CHORUS	Chorus Send
0x0081	CONN_DST_REVERB	Reverb Send

* Denotes optional 6-channel output

Modulator LFO Destinations

0x0104	CONN_DST_LFO_FREQUENCY	LFO Frequency
0x0105	CONN_DST_LFO_STARTDELAY	LFO Start Delay Time

Vibrato LFO Destinations

0x0114	CONN_DST_VIB_FREQUENCY	Vibrato Frequency
0x0115	CONN_DST_VIB_STARTDELAY	Vibrato Start Delay

EG Destinations

0x0206	CONN_DST_EG1_ATTACKTIME	EG1 Attack Time
0x0207	CONN_DST_EG1_DECAYTIME	EG1 Decay Time
0x0208	CONN_DST_EG1_RESERVED	EG1 Reserved
0x0209	CONN_DST_EG1_RELEASETIME	EG1 Release Time
0x020A	CONN_DST_EG1_SUSTAINLEVEL	EG1 Sustain Level
0x020B	CONN_DST_EG1_DELAYTIME	EG1 Delay Time
0x020C	CONN_DST_EG1_HOLDTIME	EG1 Hold Time
0x020D	CONN_DST_EG1_SHUTDOWNTIME	EG1 Shutdown Time
0x030A	CONN_DST_EG2_ATTACKTIME	EG2 Attack Time
0x030B	CONN_DST_EG2_DECAYTIME	EG2 Decay Time
0x030C	CONN_DST_EG2_RESERVED	EG2 Reserved
0x030D	CONN_DST_EG2_RELEASETIME	EG2 Release Time
0x030E	CONN_DST_EG2_SUSTAINLEVEL	EG2 Sustain Level
0x030F	CONN_DST_EG2_DELAYTIME	EG2 Delay Time
0x0310	CONN_DST_EG2_HOLDTIME	EG2 Hold Time

Filter Destinations

0x0500	CONN_DST_FILTER_CUTOFF	Filter Cutoff Frequency
0x0501	CONN_DST_FILTER_Q	Filter Resonance

Transforms

0x0000	CONN_TRN_NONE	No Transform
0x0001	CONN_TRN_CONCAVE	Concave Transform
0x0002	CONN_TRN_CONVEX	Convex Transform
0x0003	CONN_TRN_SWITCH	Switch Transform

2.11 <wlnk-ck>, Wave Link Chunk

The <wlnk-ck> specifies where the wave data can be found for an instrument region in a DLS stream. The <wlnk-ck> is defined as follows:

```
<wlnk-ck>   →   wlnk(
                <fusOptions:USHORT>
                <usPhaseGroup:USHORT>
                <ulChannel:ULONG>
                <ulTableIndex:ULONG>
                )
```

The <wlnk-ck> chunk:

Field	Description
fusOptions	Specifies flag options for this wave link. All bits not defined must be set to 0. The defined flags are as follows: **F_WAVELINK_PHASE_MASTER** 0x0001 Specifies that this link is the master in a group of phase locked wave links. **F_WAVELINK_MULTICHANNEL** 0x0002 Indicates that the *ulChannel* field provides the channel steering information and all the channel steering data in the articulation chunk should be ignored.
usPhaseGroup	Specifies a group number for samples which are phase locked. All waves in a set of wave links with the same group are phase locked and follow the wave in the group with the F_WAVELINK_PHASE_MASTER flag set. If a wave is not a member of a phase locked group, this value should be set to 0.
ulChannel	Specifies the channel placement of the sample. This is used to place mono sounds within a stereo pair or for multi-track placement. Each bit position within the *ulChannel* field specifies a channel placement with bit 0 specifying a mono sample or the left channel of a stereo file. Bit assignments are as follows:

0	Left (or mono)		10	Side Right
1	Right Channel		11	Top
2	Center		12	Top Front Left
3	Low Frequency Energy		13	Top Front Center
4	Surround Left		14	Top Front Right
5	Surround Right		15	Top Rear Left
6	Left of Center		16	Top Rear Center
7	Right of Center		17	Top Rear Right
8	Surround Center		18-31	Reserved (DO NOT USE)
9	Side Left			

ulTableIndex	Specifies the 0 based index of the cue entry in the wave pool table.

To create a group of phase-locked samples, the *usPhaseGroup* field of each <wlnk-ck> chunk must be set to the same value, which must be non-zero. Additionally, the F_WAVELINK_PHASE_MASTER bit must be set in one, and only one, of the <wlnk-ck> chunks, which will act as the master for all phase-locked samples.

When using phase-locked samples, all oscillators take their frequency from the oscillator associated with the phase-master sample. All parameters that can affect the frequency of a slaved oscillator *must be ignored*. This includes any connections with *ulDestination* set to DST_PITCH, as well as sample rate and fine tuning parameters in the <wsmp-ck> chunk. All slaved oscillators must play back at the frequency of the master oscillator and must retain sample lock to within plus or minus one half-sample at the device sample output rate.

When the F_WAVELINK_MULTICHANNEL bit is set, the channel assignment of a given sample is fixed, overriding any parameters in the articulation data. This allows for steering of data to specific speaker locations without regard to pan controls or other sends. When the F_WAVELINK_MULTICHANNEL bit is zero, channel assignment is not fixed, allowing for dynamic assignment of the channel via articulation data. Multichannel mode is optional, and Mobile DLS devices need not support it. Content intended to be cross-platform compatible *must* not use the multichannel mode, unless the <wlnk-ck> is guarded through the use of a conditional chunk to protect it from DLS-2 and Mobile DLS devices that may not support multichannel playback.

2.12 <wsmp-ck>, Wave Sample Chunk

The <wsmp-ck> wave sample chunk describes the minimum necessary information needed to allow a synthesis engine to use a <wave-list> wave file chunk. The <wsmp-ck> is defined as follows:

```
<wsmp-ck>    →          wsmp(
                        <cbSize:ULONG>
                        <usUnityNote:USHORT>
                        <sFineTune:SHORT>
                        <lGain:LONG>
                        <fulOptions:ULONG>
                        <cSampleLoops:ULONG>
                        <wavesample-loop>...
                        )

<wavesample-loop> →     struct
                        {
                            ULONG   cbSize;
                            ULONG   ulLoopType;
                            ULONG   ulLoopStart;
                            ULONG   ulLoopLength;
                        }
```

The <wsmp-ck> chunk:

Field	Description
cbSize	Specifies the size of the structure in bytes. *This size does not include the loop records.* This field is needed to distinguish the amount of data in the structure versus the list of loops and allow for additions to this structure in the future. This cannot be determined from the chunk size.
usUnityNote	Specifies the MIDI note which will replay the sample at original pitch. This value ranges from 0 to 127 (a value of 60 represents Middle C, as defined by the MIDI specification).

sFineTune Specifies the tuning offset from the usUnityNote in 16 bit relative pitch.

lGain Specifies the gain to be applied to this sample in 32 bit relative gain units.

fulOptions Specifies flag options for the digital audio sample. Current options are:

F_WSMP_NO_TRUNCATION 0x0001

This option specifies that a synthesis engine is not allowed to truncate the bit depth of the sample if it cannot synthesize at the bit depth of the digital audio. If the NO_TRUNCATION bit is set, the device is not allowed to truncate the data to a bit depth less than that of the original sample. Note that, if truncation is disallowed, the wave may take up more memory in the device and thus may fail to download due to memory constraints. If the bit is clear, the synthesis engine may truncate the bit depth, if required.

F_WSMP_NO_COMPRESSION 0x0002

This option specifies that a synthesis engine is not allowed to use compression in its internal synthesis engine for the digital audio sample. If the NO_COMPRESSION bit is set, the device is not allowed to compress the data. Note that, if compression is disallowed, the wave may take up more memory in the device and thus may fail to download due to memory constraints. If the bit is clear, the synthesis engine may compress the digital audio samples, if required.

cSampleLoops Specifies the number (count) of <wavesample-loop> records that are contained in the <wsmp-ck> chunk. The <wavesample-loop> records are stored immediately following the **cSampleLoops** data field. One shot sounds will have the **cSampleLoops** field set to 0. Looped sounds will have the **cSampleLoops** field set to 1. Values greater than 1 are not yet defined at this time.

The <wavesample-loop> structure:

Field	Description		
cbSize	Specifies the size of the structure in bytes.		
ulLoopType	Specifies the loop type:		
	WLOOP_TYPE_FORWARD	0x0000	Forward Loop
	WLOOP_TYPE_RELEASE	0x0001	Loop and Release
ulLoopStart	Specifies the start point of the loop in samples as an absolute offset from the beginning of the data in the <data-ck> subchunk of the <wave-list> wave file chunk.		
ulLoopLength	Specifies the length of the loop in samples.		

2.13 <ptbl-ck>, Pool Table Chunk

The <ptbl-ck> pool table chunk contains a list of cross-reference entries to digital audio data within the wave pool. The <ptbl-ck> is defined as follows:

```
<ptbl-ck>   →   ptbl(
                <cbSize:ULONG>
                <cCues:ULONG>
                <poolcues(s)>...
                )
```

```
<poolcue>   →        struct
                     {
                     ULONG       ulOffset;
                     }
```

The <ptbl-ck> chunk:

Field	Description
cbSize	Specifies the size of the structure in bytes. *This size does not include the poolcue records.* This field is needed to distinguish the amount of data in the structure versus the list of cues and allow for additions to this structure in the future. This cannot be determined from the chunk size.
cCues	Specifies the number (count) of <poolcue> records that are contained in the <ptbl-ck> chunk. The <poolcue> records are stored immediately following the **cCues** data field.
ulOffset	Specifies the absolute offset in bytes from the beginning of the wave pool data to the correct entry in the wave pool. By using an offset to the data of each pool entry, a synthesis engine does not have to walk from the beginning of the list to find any given set of wave data but can immediately seek to the correct location in the wave pool chunk. Additions and deletions of entries in the wave pool need only modify the pool table as opposed to modifying every <wave-link> in the instrument chunks.

2.14 <vers-ck>, Version Chunk

The <vers-ck> defines an optional version stamp within a collection. The version stamp is intended primarily as a mechanism for managing DLS resources, in particular when installing DLS files. It indicates the version of the contents of the file, *not the DLS specification level.* A setup program reads the version chunk in a previously installed DLS file to identify whether it should install a newer file over it. The version stamp follows the same four number format as the version stamps used in executable code (.exe and .dll files). It is left to the discretion of the DLS file author to assign a version number scheme to a file.

The <vers-ck> is defined as follows:

```
<vers -ck>   →        vers(
                      <dwVersionMS:DWORD>
                      <dwVersionLS:DWORD >
                      )
```

The <vers -ck> chunk:

Field	Description
dwVersionMS	Specifies the high-order 32 bits of the binary version number for the file. The value of this member is used with the value of the dwVersionLS member to form a 64-bit version number. This 32 bit value can be further broken down with HIWORD(dwVersionMS) and LOWORD(dwVersionMS) to get two 16 bit values (version info is actually 4 16 bit values.
dwVersionLS	Specifies the low-order 32 bits of the binary version number for the file. The value of this member is used with the dwVersionMS value to form a 64-bit version number. Use HIWORD(dwVersionLS) and LOWORD(dwVersionLS) to extract the sixteen bit values.

The version chunk is essentially borrowed from the dwFileVersionMS and dwFileVersionLS fields in the Microsoft® VS_FIXEDFILEINFO structure which is used to store version information in .exe and .dll files. As such, the version can be broken down into four segments: the high and low words of the MS and the high and low words of the LS. For example, "VERSION 3,10,0,61" is translated into: dwVersionMS = 0x0003000a and dwVersionLS = 0x0000003d.

A <vers-ck> version chunk is typically (but not necessarily) near the top of the enclosing DLS chunk so it can be quickly found by installation software.

2.15 <INFO-list>, INFO List Chunk

The INFO list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An INFO list is a LIST chunk with list type INFO. The following shows a sample INFO list chunk:

```
< INFO-list >   →       LIST(  'INFO'  [<info_text-ck>]...)
```

Example:

```
< INFO-list >   →       LIST(       'INFO'
                        INAM("Distorted Tuba"Z)
                         ICMT("The tuba meets Eddie Van Halen"Z)
                        )
```

An INFO list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an INFO list. Each chunk contains a ZSTR, or null-terminated text string.

The <info_text-ck> chunks include:

Chunk ID	Description
IARL	Archival Location. Indicates where the subject of the file is archived**.**
IART	Artist. Lists the artist of the original subject of the file. For example, Les Paul.
ICMS	Commissioned. Lists the name of the person or organization that commissioned the subject of the file. For example, Pope Julian II.
ICMT	Comments. Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do not include newline characters.
ICOP	Copyright. Records the copyright information for the file. For example, Copyright Encyclopedia International 1991. If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	Creation date. Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example, 1553-05-03 for May 3, 1553.

IENG	Engineer. Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example, Smith, John; Adams, Joe.
IGNR	Genre. Describes the original work, such as, jazz, classical, rock, techno, rave, neo british pop grunge metal, etc.
IKEY	Keywords. Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example, FX; death; murder.
IMED	Medium. Describes the original subject of the file, such as, record, CD, and so forth.
INAM	Name. Stores the title of the subject of the file, such as, Seattle From Above.
IPRD	Product. Specifies the name of the title the file was originally intended for, such as World Ruler V.
ISBJ	Subject. Describes the contents of the file, such as Music of the New World Order.
ISFT	Software. Identifies the name of the software package used to create the file, such as Sonic Foundry Sound Forge.
ISRC	Source. Identifies the name of the person or organization who supplied the original subject of the file. For example, Trey Research.
ISRF	Source Form. Identifies the original form of the material that was digitized, such as record, sampling CD, TV sound track, and so forth. This is not necessarily the same as IMED.
ITCH	Technician. Identifies the technician who sampled the subject file. For example, Smith, John.

2.16 DLS Wave File Format

The <wave-list> wave file chunk is defined as follows. **Programs must expect (and ignore) any unknown chunks encountered, as with all RIFF forms**. However, <fmt-ck> must always occur before <data-ck>, and both <fmt-ck> and <data-ck> chunks are mandatory in a <wave-file>.

A <wave-list> is similar to an embedded wave file, as would be generated by a wave editing program. These wave file chunks are referenced from the <rgn-list> chunks inside each <ins-list> chunk, and accessed via the <poolcue> entries in the <ptbl-ck> pool table chunk. The <wave-list> data is equivalent to that found in a normal <WAVE-form> WAV file, with two exceptions. First, the initial three DWORD values in a normal <WAVE-form> WAV file (which are 'RIFF' [size] 'WAVE') are replaced with a <wave-list> chunk header ('LIST' [size] 'wave'). Second, an optional <dlid-ck> is inserted as the first element following the <wave-list> chunk header, in order to provide a globally unique identifier for the <wave-list> wave file data.

DLS does not require the use of <fact-ck>, <cue-ck>, <playlist-ck> or <assoc-data-list> chunks; therefore, they are intentionally left out of the specification.

```
<wave-list> → LIST( 'wave' // note: 'wave' is lower case
                    [<dlid-ck>]
                    <fmt-ck>
                    [<wsmp-ck>]
                    [<fact-ck>]
                    [<cue-ck>]
                    [<playlist-ck>]
                    [<assoc-data-list>]
                    [<INFO-list>]
                    <data-ck>
                  )
```

2.16.1 Format Chunk <fmt-ck>

The WAVE format chunk <fmt-ck> specifies the format of the <wave-data>. The <fmt-ck> is defined as follows:

```
<fmt-ck>                  →     fmt ( <common-fields>
                                <format-specific-fields>
                                )

<common-fields>           →     struct      {
                                WORD        wFormatTag;
                                WORD        wChannels;
                                DWORD       dwSamplesPerSec;
                                DWORD       dwAvgBytesPerSec;
                                WORD        wBlockAlign;
                                }
```

```
<format-specific-fields>            →    <PCM-format-specific>
                                         OR
                                         <WAVE-format-extensible-specific>

<PCM-format-specific>               →    struct {
                                         WORD        wBitsPerSample;
                                         }

<WAVE-format-extensible-specific>   →    struct {
                                         WORD        wBitsPerSample;
                                         WORD        cbSize;
                                             union {
                                             WORD wValidBitsPerSample;
                                             WORD wSamplesPerBlock;
                                             WORD wReserved;
                                             }
                                         DWORD       dwChannelMask;
                                         GUID        SubFormat;
                                         }
```

The <common-fields> struct:

Field	Description
wFormatTag	A number indicating which format specific field to use (either <PCM-format-specific> or <WAVE-format-extensible-specific>). If the number is WAVE_FORMAT_EXTENSIBLE (0xFFFE), the data following the <common-fields> is in the form detailed in <WAVE-format-extensible-specific>, below; if it is any other number, the data following the <common-fields> is in the form detailed in <PCM-format-specific>, below. In the case of <PCM-format-specific>, the number indicates the encoding of the waveform data. The format WAVE_FORMAT_PCM (0x0001) is defined to be Microsoft Pulse Code Modulation (PCM) format. The MMA will maintain a registry of supported wFormatTags.
wChannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo. DLS-1 supports only mono data (value = "1").
dwSamplesPerSec	The sampling rate (in samples per second) at which each channel should be played.
dwAvgBytesPerSec	The average number of bytes per second at which the waveform data should transferred. Playback software can estimate the buffer size using this value.
wBlockAlign	The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of wBlockAlign bytes of data at a time, so the value of wBlockAlign can be used for buffer alignment.

The <PCM-format-specific> struct:

Field	Description
wBitsPerSample	Specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel. Mobile DLS supports only 8 or 16 bit samples. If this field is not needed, then it should be set to zero.

The <WAVE-format-extensible-specific> struct:

Field	Description
wBitsPerSample	Specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel. Mobile DLS supports only 8 or 16 bit samples. If this field is not needed, then it should be set to zero.
cbSize	Specifies the size in bytes of the extra information in the <WAVE-format-extensible-specific> structure not including the size of **wBitsPerSample** and **cbSize**. **cbSize** must always be set to at least 22.
wValidBitsPerSample	Specifies how many bits are used per sample. If the **wValidBitsPerSample** is less than **wBitsPerSample**, then the actual PCM data is aligned left and all extra bits are at the least significant part of the word.
wSamplesPerBlock	Specifies how many samples are stored in one compressed block. wSamplesPerBlock is used for formats with fixed number of samples per block. If wSamplesPerBlock is zero, a variable amount of samples is contained in each block of compressed audio data.
wReserved	If neither wValidBitsPerSample or wSamplesPerBlock apply to the audio data being described by the <WAVE-format-extensible-specific> structure, set the wReserved field to zero.
dwChannelMask	The field dwChannelMask indicates which channels are present in the multi-channel stream.
SubFormat	The SubFormat field is set to the GUID that specifies the type of encoded waveform data described by the <WAVE-format-extensible-specific> structure. The MMA will maintain a registry of supported GUIDs.

2.16.2 Data Chunk <data-ck>

The <data-ck> contains the waveform data. It is defined as follows:

```
<data-ck>    →          data( <wave-data> )
```

The <data-ck> chunk:

Field	Description
<wave-data>	This is the waveform data encoded in the form described above in <common-fields>, wFormatTag and <WAVE-format-extensible-specific>, SubFormat. For PCM encoded waveform data, see Data Packing for WAVE_FORMAT_PCM Files and Data Format of the WAVE_FORMAT_PCM Samples sections.

2.16.3 Data Packing for WAVE_FORMAT_PCM Files

In a single-channel <wave-file>, samples are stored consecutively. For stereo <wave-file>, channel 0 represents the left channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel <wave-file>, samples are interleaved. Only channel 0 is supported for DLS-1 and DLS-2 and Mobile DLS. Stereo file format are shown for completeness.

The following diagrams show the data packing for 8-bit mono and stereo WAVE files:

2.16.4 Data Packing for 8-Bit Mono PCM

Sample 1	Sample 2	Sample 3	Sample 4
Channel 0	Channel 0	Channel 0	Channel 0

2.16.5 Data Packing for 8-Bit Stereo PCM

Sample 1	Sample 1	Sample 2	Sample 2
Channel 0	Channel 1	Channel 0	Channel 0
(left)	(right)	(left)	(right)

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:

2.16.6 Data Packing for 16-Bit Mono PCM

Sample 1	Sample 1	Sample 2	Sample 2
Channel 0	Channel 0	Channel 0	Channel 0
low-order	High-order	low-order	high-order
byte	Byte	byte	byte

2.16.7 Data Packing for 16-Bit Stereo PCM

Sample 1	Sample 1	Sample 1	Sample 1
Channel 0	Channel 0	Channel 1	Channel 1
(left)	(left)	(right)	(right)
low-order	High-order	low-order	high-order
byte	Byte	byte	byte

2.16.8 Data Format of the WAVE_FORMAT_PCM Samples

The data format and maximum and minimums values for PCM waveform samples of various sizes are as follows:

Sample Size	Data Format	Maximum Value	Minimum Value	Midpoint Value
8 bit	Unsigned	255 (0xFF)	0	128 (0x80)
16 bit	Signed	32767 (0x7FFF)	-32768 (0x8000)	0

2.17 Instrument Object Hierarchy

The following charts show how the instrument object hierarchies described in the DLS Device Architecture specification correspond to particular chunks within the <DLS-form> file.

Object Hierarchy: DLS-1 Melodic Instrument	**Corresponding <DLS-form> chunks**
Melodic Instrument ----------	<ins-list> instrument, <insh-ck> instrument header, <INFO-list>
Global	<lart-list> list of articulators
Connection Blocks----------	<art1-ck> articulator
Region (1..16)	<lrgn-list> list of regions, holding <rgn-list> region chunks
Wave Link----------	<wlnk-ck> wave link
Wave Sample ----------	<wsmp-ck> wave sample
Wave File----------	<wave-list> wave file
Wave Sample (optional)----------	<wsmp-ck> wave sample
Wave Data----------	<data-ck>

Object Hierarchy: DLS-1 Drum Kit	**Corresponding <DLS-form> chunks**
Drum Kit----------	<ins-list> instrument, <insh-ck> instrument header, <INFO-list>
Region (1..128)----------	<lrgn-list> list of regions, holding <rgn-list> region chunks
Wave Link----------	<wlnk-ck> wave link
Wave Sample----------	<wsmp-ck> wave sample
Wave File----------	<wave-list> wave file
Wave Sample (optional)----------	<wsmp-ck> wave sample
Wave Data----------	<data-ck>
Region Articulation----------	<lart-list> list of articulators
Connection Blocks ----------	<art1-ck> articulator

Object Hierarchy: DLS-2 Instrument	**Corresponding <DLS-form> chunks**
Instrument ----------	<ins-list> instrument, <insh-ck> instrument header, <INFO-list>
Global	<lart-list>,<lar2-list> list of articulators
Connection Blocks----------	<art1-ck>,<art2-ck> articulator
Region (1..n)	<lrgn-list> list of <rgn-list>,<rgn2-list> region chunks
Wave Link----------	<wlnk-ck> wave link
Wave Sample ----------	<wsmp-ck> wave sample

Wave File----------	<wave-list> wave file
Wave Sample (optional)----------	<wsmp-ck> wave sample
Wave Data ----------	<data-ck>
Region Articulation----------	<lart-list>,<lar2-list> list of articulators
Connection Blocks----------	<art1-ck>,<art2-ck> articulator

2.18 Proprietary Chunk IDs

In addition to the use of Conditional Chunks for adding proprietary features, manufacturers of DLS devices may embed product-specific data into a DLS file using a Proprietary Chunk ID. These IDs are assigned exclusively by the MMA, in order to prevent any potential error or conflicts between manufacturers accidentally using the same ID. Manufacturers wishing to obtain their own Proprietary Chunk ID should visit the MMA web site (www.midi.org) or contact the MMA for an application form.

Proprietary Chunk IDs work as follows:

- The ID is alpha-numeric and 4 bytes in length.
- The first three digits are assigned by the MMA. Manufacturers may receive three digits of their own choosing, if available and approved by the MMA. (Some choices are reserved).
- The fourth digit is assigned by the manufacturer, and should indicate a specific product (or product group with consistent features).
- Allowable characters are A-Z and 0-9, supporting 36 different products or product groups.
- Manufacturers needing more variations may apply for an additional ID.

2.19 File Examples

2.19.1 Generic DLS-1 File

This instrument collection contains two instruments, an ocean surf sound and a flute instrument. The ocean surf is loaded in bank 1 instrument 1 and responds across the whole key range. The flute sound is loaded in bank 1 instrument 2 and consists of two regions with one region for keys from 0-63 and one region for keys from 64-127.

RIFF 'DLS '
- <vers> (1,0,0,23)
- LIST 'INFO'
  - inam "Demo Collection with Ocean Surf and Flute"
  - icop "Copyright © 1996 MIDI Manufacturers Association"
- <dlid> (globally unique identifier for entire collection)
- <colh> (2 Instruments in this collection)
- LIST 'lins'
  - LIST 'ins '
    - LIST 'INFO'
      - inam "Ocean Surf"
    - <dlid> (globally unique identifier for "Ocean Surf" instrument)
    - <insh> (1 Region, location bank 5600h instrument 1)
    - LIST 'lrgn'
      - LIST 'rgn '
        - <rgnh> (responds to all keys from 0-127, velocities from 0-127)
        - <wsmp> (specifies loop points and midi root note)
        - <wlnk> (specifies Wave #1 (index = 0) in the Pool Table)
    - LIST 'lart'
      - <art1> (specifies the DLS-1 articulation for this instrument)
  - LIST 'ins '
    - LIST 'INFO'
      - inam "My Flute"
    - <dlid> (globally unique identifier for "My Flute" instrument)
    - <insh> (2 Regions, location bank 5600h instrument 2)
    - LIST 'lrgn'
      - LIST 'rgn '
        - <rgnh> (responds to all keys from 0-63, velocities from 0-127)
        - <wsmp> (specifies loop points and midi root note)

<wlnk> (specifies Wave #2 (index= 1) in the Pool Table)

LIST ‘rgn ‘

<rgnh> (responds to all keys from 64-127, velocities from 0-127)

<wsmp> (specifies loop points and midi root note)

<wlnk> (specifies Wave #3 (index = 2) in the Pool Table)

LIST ‘lart’

<art1> (specifies the DLS-1 articulation for this instrument)

<ptbl> [3 entries, [0,offset to surf][1,offset to flute lower][2,offset to flute upper]]

LIST ‘wvpl’

LIST ‘wave’ (Ocean surf sound)

<dlid> (globally unique identifier for “Ocean Surf” wave file)

LIST ‘wave’ (Flute for lower half of keyboard)

<dlid> (globally unique identifier for lower-register flute wave file)

LIST ‘wave’ (Flute for upper half of keyboard)

<dlid> (globally unique identifier for upper-register flute wave file)

2.19.2 DLS-1 File With 3rd Party Extensions

The following example shows the exact same file but with additional **Proprietary Chunks** embedded for a more complex synthesis engine. The file is still compatible with DLS-1. The additional chunks embedded in this case are all assigned to manufacturer ID "MMA", and used as follows:

<*MMAX*> A third party effect chunk which contains parameters for overall effects which can be applied to a full collection of instruments.

<*MMA1*> A third party articulation chunk which specifies parameters which can be applied to each instrument in a collection, or each region within an instrument, depending on where the chunk is located in the DLS file structure. Since the chunk appears at different locations, the same ID may be used.

<*MMA2*> Another third party articulation chunk but intended for a different DLS device from manufacturer "MMA". Since this is the second proprietary chunk to be used at the same level, it must have a unique ID.

RIFF 'DLS '
- LIST 'INFO'
  - inam "Demo Collection with Ocean Surf and Flute"
  - icop "Copyright © 1996 MIDI Manufacturers Association"
- <dlid> (globally unique identifier for entire collection)
- <colh> (2 Instruments in this collection)
- LIST 'lins'
  - LIST 'ins '
    - LIST 'INFO'
      - inam "Ocean Surf"
    - <dlid> (globally unique identifier for "Ocean Surf" instrument)
    - <insh> (1 Region, location bank 5600h instrument 1)
    - LIST 'lrgn'
      - LIST 'rgn '
        - <rgnh> (responds to all keys from 0-127, velocities from 0-127)
        - <wsmp> (specifies loop points and midi root note)
        - <wlnk> (specifies Wave #1 (index = 0) in the Pool Table)
        - *LIST 'lart'*
        - *<MMA1> (specifies 3rd party region level articulation)*
    - LIST 'lart'
      - <art1> (specifies the DLS-1 articulation for this instrument)
      - *<MMA1> (specifies 3rd party instrument level articulation)*

LIST ‘ins ’

LIST ‘INFO’

inam “My Flute”

<dlid> (globally unique identifier for “My Flute” instrument)

<insh> (2 Regions, location bank 5600h instrument 2)

LIST ‘lrgn’

LIST ‘rgn ‘

<rgnh> (responds to all keys from 0-63, velocities from 0-127)

<wsmp> (specifies loop points and midi root note)

<wlnk> (specifies Wave #2 (index = 1) in the Pool Table)

LIST ‘lart’

<MMA2> (specifies 3rd party region level articulation device)

LIST ‘rgn ‘

<rgnh> (responds to all keys from 64-127, velocities from 0-127)

<wsmp> (specifies loop points and midi root note)

<wlnk> (specifies Wave #3 (index = 2) in the Pool Table)

LIST ‘lart’

<MMA1> (specifies 3rd party region level articulation)

LIST ‘lart’

<art1> (specifies the DLS-1 articulation for this instrument)

<MMA1> (specifies 3rd party instrument level articulation)

<ptbl> [3 entries, [0,offset to surf][1,offset to flute lower][2,offset to flute upper]]

LIST ‘wvpl’

LIST ‘wave’ (Ocean surf sound)

<dlid> (globally unique identifier for “Ocean Surf” wave file)

LIST ‘wave’ (Flute for lower half of keyboard)

<dlid> (globally unique identifier for lower-register flute wave file)

LIST ‘wave’ (Flute for upper half of keyboard)

<dlid> (globally unique identifier for upper-register flute wave file)

<MMAX> (specifies 3rd party effects for the complete collection)

2.19.3 Generic DLS-2 File

This instrument collection contains a Strings and Bell layer instrument. The strings stretch from MIDI note 21 to 108, and contain two velocity splits, while the bell layer plays only from MIDI note 60 to 108.

RIFF 'DLS '
- <vers> (2,1,3,15)
- LIST 'INFO'
  - inam "Demo Strings plus Bell Pad"
  - icop "Copyright © 1998 MIDI Manufacturers Association"
- <dlid> (globally unique identifier for entire collection)
- <colh> (1 Instrument in this collection)
- LIST 'lins'
  - LIST 'ins '
    - LIST 'INFO'
      - inam "Strings plus Bell Pad"
    - <dlid> (globally unique identifier for "Strings plus Bell Pad" instrument)
    - <insh> (3 Regions, location bank 0100h instrument 1)
    - LIST 'lar2'
      - <art2> (specifies the DLS-2 articulation for this instrument)
    - LIST 'lrgn'
      - LIST 'rgn2'
        - <rgnh> (responds to all keys from 21-108, velocities from 0-63)
        - <wsmp> (specifies loop points and midi root note)
        - <wlnk> (specifies Wave #1 (index = 0) in the Pool Table)
        - LIST 'lar2'
          - <art2> (specifies the DLS-2 articulation for this region)
      - LIST 'rgn2'
        - <rgnh> (responds to all keys from 21-108, velocities from 64-127)
        - <wsmp> (specifies loop points and midi root note)
        - <wlnk> (specifies Wave #1 (index = 0) in the Pool Table)
        - LIST 'lar2'
          - <art2> (specifies the DLS-2 articulation for this region)
- <ptbl> [3 entries, [0,offset to strings vel 0-63][1, offset to strings vel 64-127][2,offset to bell]]
- LIST 'wvpl'
  - LIST 'wave' (String velocity layer 0-63)
    - <dlid> (globally unique identifier for "Strings" layer 0-63 wave file)

LIST ‘wave’ (String velocity layer 64-127)

<dlid> (globally unique identifier for “Strings” layer 64-127 wave file)

LIST ‘wave’ (Bell sample for layer)

<dlid> (globally unique identifier for “Bell” layer wave file)

3. Compatibility Notes

3.1 Coding Requirements and Recommendations

When writing code to read and write DLS files, it is imperative that the programmer takes into account the following:

- Since this is an expandable format, DLS file parsers must be able to ignore and correctly skip unrecognized chunks in a DLS file. This is standard practice for RIFF files but is extremely important with the DLS format since one of its purposes is to allow manufacturers to embed proprietary chunks which exploit advanced features of their synthesis engines.
- Ordering of chunks within the file format should not be assumed. For instance, a programmer should not assume that within a <rgn-list> instrument region, chunks will always appear in the order of <rgnh-ck>,<wsmp-ck>,<wlnk-ck>, <lart-list>. They may actually appear in any order within the list.
- If a <wsmp-ck> does not exist in a region list, the synthesis engine will use the <wsmp-ck> found in the referenced wave data. If a <wsmp-ck> exists in a region list, its values are used instead of any found in the referenced wave data. If a <wsmp-ck> exists in neither the region list nor the referenced data, then default values are used. These are One shot Sound, with a MIDI Unity note of 60 (Middle C), no attenuation, and no fine tune.
- Connections that are implied by the DLS device architecture must not be written to the DLS file. Implied connections include: MIDI Note to Key, RPN2 to Key, MIDI Controller 7 to Gain, MIDI Controller 11 to Gain, MIDI Controller 10 to Pan, EG1 to Attenuation, Pitch Wheel RPN0 to Pitch.
- Applications modifying the voice allocation mode should reset the DLS device to the default (static MIDI channel priority) upon exiting to insure predictable polyphony.

3.2 DLS-1 and DLS-2 Compatibility

The following restrictions are placed upon the file format and must be followed for DLS-1 and DLS-2 compatibility:

- For the <wlnk-ck> wave link the only allowed type for the **ulChannel** field is WAVELINK_CHANNEL_LEFT. This means that for a DLS-1 file only mono sound data is allowed. **fusOptions, usPhase Group** and other values for **ulChannel** other than **ulChannel** = WAVELINK_CHANNEL_LEFT should be set to default values since these fields are optional and are not required to be supported in DLS-1.
- For the embedded wave files <wave-list> within the <wvpl-list> the only data format allowed is WAVE_FORMAT_PCM. The files also must be 8 or 16 bit mono files. This means that only mono files which are 8 or 16 bit PCM data are allowed for DLS-1 and DLS-2. See Section 5 for more details on the wave format.
- For the <rgnh-ck> the allowed key groupings for the **ulKeyGroup** field are 0 for no key grouping and 1-15 for key groupings. This means there are 15 allowable key groupings within an instrument.
- Some DLS devices support playback rates of only 22.05 kHz. To prevent samples from aliasing on these devices, it is recommended that samples for these devices be restricted to 22.05 kHz or below.
- 0dB is the defined as the maximum output level at the Volume summing node (DST_GAIN, known as DST_ATTENUATION in DLS-1). If a sound designer exceeds 0dB, the output level will be clipped to 0dB.

3.3 DLS-1 Compatibility

The following restrictions are placed upon the file format and must be followed for DLS-1 compatibility. Additional information on how to code a mixed DLS-1 and DLS-2 file may be found in the section on Conditional Chunks.

- For the <lrgn-list> a maximum of 16 regions can exist for any given melodic instrument and 128 regions for a drum instrument.
- The *RangeVelocity* field is not supported by the DLS-1 specification, so all regions' *RangeVelocity* fields for a DLS-1 collection should be set to 0 for the *usLow* field and 127 for the *usHigh* field.
- DLS-1 regions are not allowed to overlap. This means that "splits" are supported by DLS-1 but "layers" are not. Therefore, an instrument's regions *RangeKey* field can not overlap.
- DLS-1 melodic instruments may have a single articulation data list <lart-list> at the instrument level, which is the global articulation data for all regions.
- DLS-1 drum instruments may have local articulation data lists <lart-list> for each region, but may not have global articulation data at the instrument level.
- To detect whether a given instrument is a drum or melodic instrument, a program should check the instrument header's **ulBank** field bit 31. If ulBank bit 31 is equal to 1, then the instrument is a drum instrument. If **ulBank** bit 31 is equal to 0, then the instrument is a melodic instrument. The distinction between drum and melodic instruments will cease to exist in a future revision of the DLS specification.
- For the <art1-ck> the **ulDefaultPan** field is only used when specifying articulation data for regions of Drum instruments. For melodic instruments this value should not be specified in the connection list.
- While all DLS-1 devices will support the full +2/-4 octave transposition range, certain DLS-1 devices have other limits on the transposition range. This limitation is imposed by available memory bandwidth and is determined by the number of voices playing and the amount of transposition occurring on each voice. The following formula may be applied to ensure that problems are not encountered with such devices: For each active voice, divide the playback frequency by the frequency at which the sample was recorded. The sum of the ratios of all active voices must be less than or equal to 36. For example, if you have 24 voices playing and each sample is being transposed up 7 semitones (~1.5 times the original frequency), the sum is 24 * 1.5 = 36. However, if one of those voices was transposing up an octave (2 times the original frequency), the sum would be 23 * 1.5 + 2 = 36.5, which would exceed the limitation of some DLS-1 devices. Note that when calculating the transposition, you must consider the transposition that occurs as the sample is transposed from its root key, tuning fields in the sample header and articulation data, the Fine Tuning RPN, and any effects from Pitch Bend.
- DLS-1 Devices do not support the use of the Coarse Tuning RPN on MIDI Channel 10.

3.4 Mod LFO to Gain Change (DLS 2.1)

Due to ambiguities in the DLS-1 specification, there were discrepancies in the implementation of the Mod LFO to Gain connection. The majority of DLS-1 synthesizers that have been deployed implement this connection as bipolar, non-inverted. However, the initial release of the DLS-2 specification requires a unipolar, inverted connection for Mod LFO to Gain, as well as for the Mod LFO CC1 to Gain and Mod LFO Channel Pressure to Gain connections. At the time, there had been no significant deployment of DLS-2 synthesizers, therefore, to facilitate backward compatibility, the DLS-2 specification was revised to reflect the practices established by the majority of DLS-1 synthesizers. The DLS-2.1 implementation of Mod LFO to Gain resulted in an increase of signal amplitude during the initial phase of the LFO, assuming a positive lScale value.

3.5 DLSID Integrity

The following requirements must be observed to maintain the integrity of DLSID Globally Unique Identifiers:

Support for DLSID Identifiers is optional, not required. However, a DLS implementation which does not support the use of DLSID Identifiers or the generation of new DLSID values must still observe several requirements. First, such implementations must be prepared to ignore and skip over any DLSIDs found within a DLS file, just as if the <dlid-ck> DLSID were an unrecognized chunk. Second, if such an implementation ***modifies*** a DLS file, it **must** remove

any <dlid-ck> chunk found within the top-level <DLS-form> and any additional <dlid-ck> chunks found within the modified portions of the DLS file.

Leaving the unchanged <dlid-ck> chunks within the modified file would violate the DLSID "uniqueness" rule. Finally, when exporting a DLS wave file from a <DLS-form> collection to a raw <WAVE-form> wave file, any embedded <dlid-ck> chunks should be stripped out.

For implementations which **do** support the use of <dlid> DLSID chunks:

If an object has a DLSID associated with it, a new DLSID value *must* be generated whenever the associated object is modified. This includes changes to an <INFO-list> chunk. If an internal object, such as an instrument or wave is changed, one must change the DLSID for *both* the object (instrument or wave) *and* the enclosing <DLS-form> collection Note that if <vers-ck> version chunks and <dlid-ck> identifiers are both used in the same file, changing the <vers-ck> requires a concurrent change to all associated <dlid-ck> values.

A DLS-aware utility which is importing a DLS object from one <DLS-form> collection into another <DLS-form> collection should not change the DLSID of the object being copied. In other words, if one is copying a <ins-list> or <wave-list> chunk from one <DLS-form> into another, one should *not* change the <dlid-ck> within the <ins-list> or <wave-list>, since this data is not being changed. Of course, one *must* change the collection DISID (the top-level <dlid-ck> within the target <DLS-form>), because importing an object into the collection changes that collection.

Wave files need special handling, since they are often generated or processed by applications which are not aware of DLSID chunks. A ***raw*** wave file is a stand-alone <WAVE-form> file, distinguished by the initial sequence 'RIFF'[size]'WAVE'. A ***DLS*** wave file is an embedded <wave-list> chunk, normally kept inside a <DLS-form> collection, distinguished by the initial sequence 'LIST'[size]'wave' (upper case vs. lower case is significant).

When importing a raw <WAVE-form> wave file into a <DLS-form> collection, generate and insert a new <dlid-ck> for the wave file (replacing any pre-existing <dlid-ck> within the raw wave file).

When exporting a DLS wave file from a <DLS-form> collection to a raw <WAVE-form> wave file, the embedded <dlid-ck> should be stripped out.

FROM	TO	Comment
<DLS-form>	<DLS-form>	Keep DLSID for instrument or wave file (data is unchanged) Change DLSID for target <DLS-form>
<WAVE-form> (raw wave file)	<DLS-form>	Remove pre-existing DLSID from raw wave file, if any Generate new DLSID for wave file object within DLS collection.
<DLS-form>	<WAVE-form>	Remove DLSID during export

It is recommended that each <dlid-ck> DLSID chunk should be placed as the *first* chunk within the enclosing LIST or FORM chunk, in order to facilitate searching. However, this ordering should not be assumed.

4. DLS Header Files

The following header files include defines and structures for DLS-1 and DLS-2 synthesizers. The DLS-2 header file augments the DLS-1 header file, and both must be included in any DLS-2 synthesizer implementation.

4.1 DLS-1 Header File

```
/*

    dls.h

    Description:

    Interface defines and structures for the Instrument Collection Form
    RIFF DLS.

    Written by Sonic Foundry 1996.  Released for public use.

*/

#ifndef _INC_DLS
#define _INC_DLS

/*

   Layout of an instrument collection:


   RIFF [] 'DLS ' [colh,INSTLIST,WAVEPOOL,INFOLIST]

   INSTLIST
   LIST [] 'lins'
                LIST [] 'ins ' [insh,RGNLIST,ARTLIST,INFOLIST]
                LIST [] 'ins ' [insh,RGNLIST,ARTLIST,INFOLIST]
                LIST [] 'ins ' [insh,RGNLIST,ARTLIST,INFOLIST]

   RGNLIST
   LIST [] 'lrgn'
                LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]
```

```
                LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]
                LIST [] 'rgn '  [rgnh,wsmp,wlnk,ARTLIST]

   ARTLIST
   LIST [] 'lart'
           'art1' level 1 Articulation connection graph
           'art2' level 2 Articulation connection graph
           '3rd1' Possible 3rd party articulation structure 1
           '3rd2' Possible 3rd party articulation structure 2 .... and so on

   WAVEPOOL
   ptbl [] [pool table]
   LIST [] 'wvpl'
                [path],
                [path],
                LIST [] 'wave',RIFFWAVE
                LIST [] 'wave',RIFFWAVE
                LIST [] 'wave',RIFFWAVE
                LIST [] 'wave',RIFFWAVE
                LIST [] 'wave',RIFFWAVE

   INFOLIST
   LIST [] 'INFO'
                'icmt' 'One of those crazy comments.'
                'icop' 'Copyright (C) 1996 Sonic Foundry'

*/


/*
   FOURCC's used in the DLS file
*/

#define FOURCC_DLS   mmioFOURCC('D','L','S',' ')
#define FOURCC_COLH  mmioFOURCC('c','o','l','h')
#define FOURCC_WVPL  mmioFOURCC('w','v','p','l')
#define FOURCC_PTBL  mmioFOURCC('p','t','b','l')
#define FOURCC_PATH  mmioFOURCC('p','a','t','h')
#define FOURCC_wave  mmioFOURCC('w','a','v','e')
#define FOURCC_LINS  mmioFOURCC('l','i','n','s')
```

```
#define FOURCC_INS   mmioFOURCC('i','n','s',' ')
#define FOURCC_INSH  mmioFOURCC('i','n','s','h')
#define FOURCC_LRGN  mmioFOURCC('l','r','g','n')
#define FOURCC_RGN   mmioFOURCC('r','g','n',' ')
#define FOURCC_RGNH  mmioFOURCC('r','g','n','h')
#define FOURCC_LART  mmioFOURCC('l','a','r','t')
#define FOURCC_ART1  mmioFOURCC('a','r','t','1')
#define FOURCC_WLNK  mmioFOURCC('w','l','n','k')
#define FOURCC_WSMP  mmioFOURCC('w','s','m','p')
#define FOURCC_VERS  mmioFOURCC('v','e','r','s')

/*
   Articulation connection graph definitions
*/

/* Generic Sources */
#define CONN_SRC_NONE             0x0000
#define CONN_SRC_LFO              0x0001
#define CONN_SRC_KEYONVELOCITY    0x0002
#define CONN_SRC_KEYNUMBER        0x0003
#define CONN_SRC_EG1              0x0004
#define CONN_SRC_EG2              0x0005
#define CONN_SRC_PITCHWHEEL       0x0006

/* Midi Controllers 0-127 */
#define CONN_SRC_CC1              0x0081
#define CONN_SRC_CC7              0x0087
#define CONN_SRC_CC10             0x008a
#define CONN_SRC_CC11             0x008b

/* Registered Parameter Numbers */
#define CONN_SRC_RPN0            0x0100
#define CONN_SRC_RPN1            0x0101
#define CONN_SRC_RPN2            0x0102

/* Generic Destinations */
#define CONN_DST_NONE             0x0000
#define CONN_DST_ATTENUATION      0x0001
#define CONN_DST_RESERVED         0x0002
#define CONN_DST_PITCH            0x0003
```

```
#define CONN_DST_PAN               0x0004

/* LFO Destinations */
#define CONN_DST_LFO_FREQUENCY     0x0104
#define CONN_DST_LFO_STARTDELAY    0x0105

/* EG1 Destinations */
#define CONN_DST_EG1_ATTACKTIME    0x0206
#define CONN_DST_EG1_DECAYTIME     0x0207
#define CONN_DST_EG1_RESERVED      0x0208
#define CONN_DST_EG1_RELEASETIME   0x0209
#define CONN_DST_EG1_SUSTAINLEVEL  0x020a

/* EG2 Destinations */
#define CONN_DST_EG2_ATTACKTIME    0x030a
#define CONN_DST_EG2_DECAYTIME     0x030b
#define CONN_DST_EG2_RESERVED      0x030c
#define CONN_DST_EG2_RELEASETIME   0x030d
#define CONN_DST_EG2_SUSTAINLEVEL  0x030e

#define CONN_TRN_NONE              0x0000
#define CONN_TRN_CONCAVE           0x0001

typedef struct _DLSVERSION {
  DWORD    dwVersionMS;
  DWORD    dwVersionLS;
}DLSVERSION, FAR *LPDLSVERSION;


typedef struct _CONNECTION {
  USHORT   usSource;
  USHORT   usControl;
  USHORT   usDestination;
  USHORT   usTransform;
  LONG     lScale;
  }CONNECTION, FAR *LPCONNECTION;


/* Level 1 Articulation Data */
```

```
typedef struct _CONNECTIONLIST {
  ULONG    cbSize;            /* size of the connection list structure */
  ULONG    cConnections;      /* count of connections in the list */
  } CONNECTIONLIST, FAR *LPCONNECTIONLIST;



/*
   Generic type defines for regions and instruments
*/

typedef struct _RGNRANGE {
  USHORT usLow;
  USHORT usHigh;
}RGNRANGE, FAR * LPRGNRANGE;

#define F_INSTRUMENT_DRUMS      0x80000000

typedef struct _MIDILOCALE {
  ULONG ulBank;
  ULONG ulInstrument;
}MIDILOCALE, FAR *LPMIDILOCALE;

/*
   Header structures found in an DLS file for collection, instruments, and
   regions.
*/

#define F_RGN_OPTION_SELFNONEXCLUSIVE  0x0001

typedef struct _RGNHEADER {
  RGNRANGE RangeKey;            /* Key range */
  RGNRANGE RangeVelocity;       /* Velocity Range */
  USHORT   fusOptions;          /* Synthesis options for this range */
  USHORT   usKeyGroup;          /* Key grouping for non simultaneous play
                                   0 = no group, 1 up is group
                                   for Level 1 only groups 1-15 are allowed */
}RGNHEADER, FAR *LPRGNHEADER;

typedef struct _INSTHEADER {
```

```
  ULONG      cRegions;          /* Count of regions in this instrument */
  MIDILOCALE Locale;            /* Intended MIDI locale of this instrument */
}INSTHEADER, FAR *LPINSTHEADER;

typedef struct _DLSHEADER {
  ULONG      cInstruments;      /* Count of instruments in the collection */
}DLSHEADER, FAR *LPDLSHEADER;

/*
   definitions for the Wave link structure
*/

/*****  For level 1 only WAVELINK_CHANNEL_MONO is valid  ****
  ulChannel allows for up to 32 channels of audio with each bit position
  specifiying a channel of playback */

#define WAVELINK_CHANNEL_LEFT    0x0001l
#define WAVELINK_CHANNEL_RIGHT   0x0002l

#define F_WAVELINK_PHASE_MASTER  0x0001

typedef struct _WAVELINK { /* any paths or links are stored right after struct
*/
  USHORT   fusOptions;     /* options flags for this wave */
  USHORT   usPhaseGroup;   /* Phase grouping for locking channels */
  ULONG    ulChannel;      /* channel placement */
  ULONG    ulTableIndex;   /* index into the wave pool table, 0 based */
}WAVELINK, FAR *LPWAVELINK;

#define POOL_CUE_NULL  0xffffffffl

typedef struct _POOLCUE {
  ULONG    ulOffset;       /* Offset to the entry in the list */
}POOLCUE, FAR *LPPOOLCUE;

typedef struct _POOLTABLE {
  ULONG    cbSize;            /* size of the pool table structure */
  ULONG    cCues;             /* count of cues in the list */
  } POOLTABLE, FAR *LPPOOLTABLE;
```

```
/*
   Structures for the "wsmp" chunk
*/

#define F_WSMP_NO_TRUNCATION     0x0001l
#define F_WSMP_NO_COMPRESSION    0x0002l


typedef struct _rwsmp {
  ULONG   cbSize;
  USHORT  usUnityNote;         /* MIDI Unity Playback Note */
  SHORT   sFineTune;           /* Fine Tune in log tuning */
  LONG    lAttenuation;        /* Overall Attenuation to be applied to data */
  ULONG   fulOptions;          /* Flag options */
  ULONG   cSampleLoops;        /* Count of Sample loops, 0 loops is one shot */
  } WSMPL, FAR *LPWSMPL;


/* This loop type is a normal forward playing loop which is continually
   played until the envelope reaches an off threshold in the release
   portion of the volume envelope */

#define WLOOP_TYPE_FORWARD   0

typedef struct _rloop {
  ULONG cbSize;
  ULONG ulType;              /* Loop Type */
  ULONG ulStart;             /* Start of loop in samples */
  ULONG ulLength;            /* Length of loop in samples */
} WLOOP, FAR *LPWLOOP;

#endif /* _INC_DLS */
```

4.2 DLS-2 Header File

```
/*

     dls2.h

     Description:
     Interface defines and structures for the DLS2 extensions of DLS.

     Written by Microsoft 1998.  Released for public use.

*/

#ifndef _INC_DLS2
#define _INC_DLS2

/*
     FOURCC's used in the DLS2 file, in addition to DLS1 chunks
*/

#define FOURCC_RGN2  mmioFOURCC('r','g','n','2')
#define FOURCC_LAR2  mmioFOURCC('l','a','r','2')
#define FOURCC_ART2  mmioFOURCC('a','r','t','2')
#define FOURCC_CDL   mmioFOURCC('c','d','l',' ')
#define FOURCC_DLID            mmioFOURCC('d','l','i','d')

/*
     Articulation connection graph definitions. These are in addition to
     the definitions in the DLS1 header.
*/

/* Generic Sources (in addition to DLS1 sources. */
#define CONN_SRC_POLYPRESSURE             0x0007      /* Polyphonic Pressure */
#define CONN_SRC_CHANNELPRESSURE                      0x0008      /* Channel
Pressure */
#define CONN_SRC_VIBRATO                              0x0009      /* Vibrato LFO
*/
#define CONN_SRC_MONOPRESSURE                         0x000a      /* MIDI Mono
pressure */
```

```
/* Midi Controllers */
#define CONN_SRC_CC91                           0x00db      /* Reverb Send
*/
#define CONN_SRC_CC93                           0x00dd      /* Chorus Send
*/



/* Generic Destinations */
#define CONN_DST_GAIN                           0x0001      /* Same as
CONN_DST_ ATTENUATION */
#define CONN_DST_KEYNUMBER              0x0005      /* Key Number Generator */


/* Audio Channel Output Destinations */
#define CONN_DST_LEFT                           0x0010      /* Left Channel
Send */
#define CONN_DST_RIGHT                          0x0011      /* Right
Channel Send */
#define CONN_DST_CENTER                         0x0012      /* Center
Channel Send */
#define CONN_DST_LEFTREAR               0x0013      /* Left Rear Channel Send
*/
#define CONN_DST_RIGHTREAR              0x0014      /* Right Rear Channel Send
*/
#define CONN_DST_LFE_CHANNEL            0x0015      /* LFE Channel Send */
#define CONN_DST_CHORUS                         0x0080      /* Chorus Send
*/
#define CONN_DST_REVERB                         0x0081      /* Reverb Send
*/


/* Vibrato LFO Destinations */
#define CONN_DST_VIB_FREQUENCY                  0x0114      /* Vibrato
Frequency */
#define CONN_DST_VIB_STARTDELAY                 0x0115      /* Vibrato
Start Delay */


/* EG1 Destinations */
#define CONN_DST_EG1_DELAYTIME                  0x020B      /* EG1 Delay
Time */
#define CONN_DST_EG1_HOLDTIME           0x020C      /* EG1 Hold Time */
#define CONN_DST_EG1_SHUTDOWNTIME       0x020D      /* EG1 Shutdown Time */



/*   EG2 Destinations */
```

```
#define CONN_DST_EG2_DELAYTIME                          0x030F      /* EG2 Delay
Time */
#define CONN_DST_EG2_HOLDTIME               0x0310      /* EG2 Hold Time */



/* Filter Destinations */
#define CONN_DST_FILTER_CUTOFF                          0x0500      /* Filter
Cutoff Frequency */
#define CONN_DST_FILTER_Q                               0x0501      /* Filter
Resonance */



/* Transforms */
#define CONN_TRN_CONVEX                                 0x0002      /* Convex
Transform */
#define CONN_TRN_SWITCH                                 0x0003      /* Switch
Transform */



/*   Conditional chunk operators */
#define DLS_CDL_AND                                     0x0001      /* X = X & Y */
#define DLS_CDL_OR                                      0x0002      /* X = X | Y */
#define DLS_CDL_XOR                                     0x0003      /* X = X ^ Y */
#define DLS_CDL_ADD                                     0x0004      /* X = X + Y */
#define DLS_CDL_SUBTRACT                  0x0005      /* X = X - Y */
#define DLS_CDL_MULTIPLY                  0x0006      /* X = X * Y */
#define DLS_CDL_DIVIDE                    0x0007      /* X = X / Y */
#define DLS_CDL_LOGICAL_AND   0x0008      /* X = X && Y */
#define DLS_CDL_LOGICAL_OR                0x0009      /* X = X || Y */
#define DLS_CDL_LT                                      0x000A      /* X = (X < Y)
*/
#define DLS_CDL_LE                                      0x000B      /* X = (X <= Y)
*/
#define DLS_CDL_GT                                      0x000C      /* X = (X > Y)
*/
#define DLS_CDL_GE                                      0x000D      /* X = (X >= Y)
*/
#define DLS_CDL_EQ                                      0x000E      /* X = (X == Y)
*/
#define DLS_CDL_NOT                                     0x000F      /* X = !X */
#define DLS_CDL_CONST                     0x0010      /* 32-bit constant */
#define DLS_CDL_QUERY                     0x0011      /* 32-bit value returned
from query */
```

```
#define DLS_CDL_QUERYSUPPORTED            0x0012      /* 32-bit value returned
from query */


/*
  Loop and Release
*/


#define WLOOP_TYPE_RELEASE 1


/*
  DLSID queries for <cdl-ck>
*/


DEFINE_DLSID(DLSID_GMInHardware, 0x178f2f24, 0xc364, 0x11d1, 0xa7, 0x60, 0x00,
0x00, 0xf8, 0x75, 0xac, 0x12);

DEFINE_DLSID(DLSID_GSInHardware, 0x178f2f25, 0xc364, 0x11d1, 0xa7, 0x60, 0x00,
0x00, 0xf8, 0x75, 0xac, 0x12);

DEFINE_DLSID(DLSID_XGInHardware, 0x178f2f26, 0xc364, 0x11d1, 0xa7, 0x60, 0x00,
0x00, 0xf8, 0x75, 0xac, 0x12);

DEFINE_DLSID(DLSID_SupportsDLS1, 0x178f2f27, 0xc364, 0x11d1, 0xa7, 0x60, 0x00,
0x00, 0xf8, 0x75, 0xac, 0x12);

DEFINE_DLSID(DLSID_SupportsDLS2, 0xf14599e5, 0x4689, 0x11d2, 0xaf, 0xa6, 0x0,
0xaa, 0x0, 0x24, 0xd8, 0xb6);

DEFINE_DLSID(DLSID_SampleMemorySize, 0x178f2f28, 0xc364, 0x11d1, 0xa7, 0x60,
0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);

DEFINE_DLSID(DLSID_ManufacturersID, 0xb03e1181, 0x8095, 0x11d2, 0xa1, 0xef, 0x0,
0x60, 0x8, 0x33, 0xdb, 0xd8);

DEFINE_DLSID(DLSID_ProductID, 0xb03e1182, 0x8095, 0x11d2, 0xa1, 0xef, 0x0, 0x60,
0x8, 0x33, 0xdb, 0xd8);

DEFINE_DLSID(DLSID_SamplePlaybackRate, 0x2a91f713, 0xa4bf, 0x11d2, 0xbb, 0xdf, 0x0,
0x60, 0x8, 0x33, 0xdb, 0xd8);


#endif            /* _INC_DLS2 */
```

4.3 Mobile DLS Header File

```
/*

        mdls.h

        Description:
        Interface defines and structures for the Mobile DLS extensions of DLS.

     Written by Creative Technology, Ltd., 2003.  Released for public use.

*/

#ifndef _INC_MDLS
#define _INC_MDLS

// {3104088D-2FC6-4a6a-A6C7-8D53035AFE1A}
DEFINE_DLSID(DLSID_SupportsMobileDLS, 0x3104088d, 0x2fc6, 0x4a6a, 0xa6,
0xc7, 0x8d, 0x53, 0x3, 0x5a, 0xfe, 0x1a);

// {ED8E8A37-97F9-46cd-922F-6DD436F8DB8D}
DEFINE_DLSID(DLSID_SupportsMobileDLSOptionalBlocks, 0xed8e8a37, 0x97f9,
0x46cd, 0x92, 0x2f, 0x6d, 0xd4, 0x36, 0xf8, 0xdb, 0x8d);

#endif  /* _INC_MDLS */
```

5. References

[1] The Complete MIDI 1.0 Detailed Specification, Document version 96.1, The MIDI Manufacturers Association, Los Angeles, CA, USA, 1996.

[2] Downloadable Sounds Level 1, Version 1.1a. The MIDI Manufacturers Association, Los Angeles, CA, USA, 1999.

[3] Downloadable Sounds Level 2.1, Version 1.0. The MIDI Manufacturers Association, Los Angeles, CA, USA, 2000.

[4] General MIDI System Level 1 (Recommended Practice RP-003), in The Complete MIDI 1.0 Detailed Specification, Document version 96.1, MIDI Manufacturers Association, Los Angeles, CA, USA, 1996.

[5] General MIDI Level 2 Specification (Recommended Practice RP-024), MIDI Manufacturers Association, Los Angeles, CA, USA, 1999.

[6] Scalable Polyphony MIDI (Recommended Practice RP-034) and 5–24 Note Device Profile For 3GPP, (Recommended Practice RP-035), MIDI Manufacturers Association, Los Angeles, CA, USA, 2002

[7] Mobile XMF Content Format Specification (Recommended Practice RP-042), MIDI Manufacturers Association, Los Angeles, CA, USA, 2004.